▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación 7

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 7

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

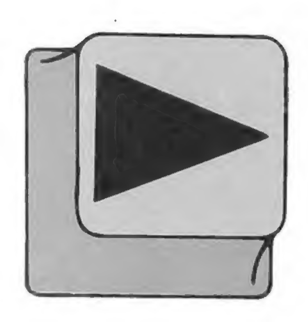
Director-editor:
RICARDO ESPAÑOL CRESPO.
Gerente:
ANTONIO G. CUERPO.
Directora de producción:
MARIA LUISA SUAREZ PEREZ.
Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.
Diseño y maquetación:
BRAVO-LOFISH.
Fotografía:
EQUIPO GALATA.
Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomuniación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: Jesús Bocho, Licenciado en Informática. LOGO: Cristina Manzanero, Licenciada en Informática. APLICACIONES: Fernando Suero, Diplomado en Telecomunicación. OTROS LENGUAJES (Sistemas operativos): Domingo Villaseñor, Diplomado en Informática, y Lenguaje C: Enrique Serrano, Ingeniero en Telecomunicación.

Ediciones Siglo Cultural, S.A.
Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.
Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.
Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.
Distribución en Ecuador: Muñoz Hnos.
Distribución en Perú: DISELPESA.
Distribución en Chile: Alfa Ltda.
Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-079-0.
ISBN de la obra: 84-7688-068-7
Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.
Imprime:
MATEU CROMO. Pinto (Madrid).

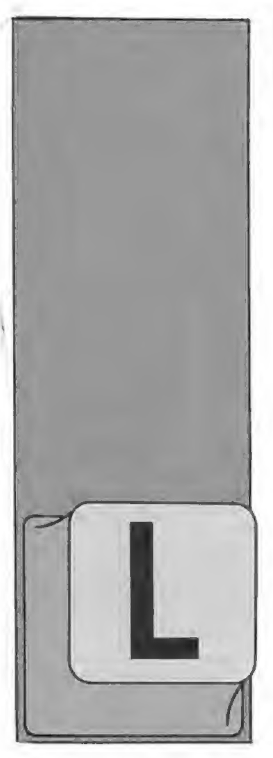
Depósito legal: M. 5.677-1987
Printed in Spain - Impreso en España.
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.
Mayo, 1987.
P.V.P. Canarias: 335,–.



INDICE

BASIC

Entrada de datos: Input



LA instrucción INPUT nos permite asignar un contenido a una variable desde el teclado. Un primer formato general sería el siguiente:

INPUT < lista de variables>

donde *la lista de variables* se compone por tantas variables como datos deseemos introducir posteriormente desde el teclado al ejecutar el programa. Generalmente todas estas variables se separan entre sí con comas. Lógicamente, si los datos que vamos a introducir son de tipo numérico tendremos que emplear variables numéricas, mientras que si son de tipo alfanumérico (cadenas), emplearemos variables alfanuméricas. En cualquier caso, siempre podemos utilizar ambos tipos de variables en un mismo INPUT si estamos manejando datos de los dos tipôs.

Veamos un primer ejemplo. El programa 1 es una modificación del programa 3 de modo que resulta mucho más general, ya que permite calcular el área y el perímetro de una circunferencia de cualquier radio.

```
10 CLS
20 INPUT R
30 LET AREA=3.1416*R^2
40 LET PERIM=2*3.1416*R
50 PRINT "RADIO=";R
60 PRINT "AREA=";AREA
70 PRINT "PERIMETRO=";PERIM
```

Al ejecutar el programa (RUN) lo primero que hace el ordenador es borrar la pantalla gracias a la instrucción CLS de la línea 10 (si nuestro ordenador es un COMMODORE, la instrucción para borrar la pantalla es 10 PRINT CHR$(147). A continuación aparece en pantalla una interrogación, que indica que el ordenador está esperando un dato. En este momento podemos introducir el dato que deseemos, que en este caso tendrá que ser de tipo numérico, ya que le vamos a suministrar el radio de una circunferencia (y, por tanto, hemos empleado una variable numérica). Después de teclear el dato debemos pulsar la tecla INTRO (o su equivalente) para que el ordenador almacene dicho dato en memoria y continúe la ejecución del programa, que finalizará con la impresión en pantalla de los resultados.

Como podemos ver, la instrucción INPUT presenta una gran ventaja, ya que cada vez que ejecutemos el programa podremos asignar un valor distinto al radio.

Sin embargo, todavía encontramos un inconveniente. ¿Qué sucedería si hiciéramos un programa con varios INPUT, o con un solo INPUT pero con varias variables? Probablemente no sabríamos qué dato nos está pidiendo en cada momento o en qué orden debemos introducirlos.

Para solucionar este problema podemos conseguir que el ordenador diga lo que pide poniendo mensajes en el INPUT. De este modo tenemos un nuevo formato:

INPUT "mensaje"; <lista de variables>

Probemos a sustituir la línea 20 del programa 1 por la línea siguiente:

```
20 INPUT "DIME EL RADIO";R
```

Al ejecutar ahora el programa aparece en pantalla el mensaje encerrado entre comillas, entonces podemos teclear el valor que deseamos asignar a la variable R (y pulsar INTRO). El resto del programa evoluciona igual que antes.

Veamos ahora un ejemplo para introducir desde el teclado cadenas de caracteres. El programa 2 nos permite introducir tres datos alfanuméricos y uno numérico.

```
10 CLS
20 INPUT "MARCA DEL VEHICULO ";V$
30 INPUT "MATRICULA ";M$
40 INPUT "COLOR ";C$
50 INPUT "ANTIGUEDAD ";A
60 CLS
70 PRINT TAB(12);"DATOS DEL VEHICULO"
80 PRINT
90 PRINT "MARCA:",V$
100 PRINT "MATRICULA:",M$
110 PRINT "COLOR:",C$
120 PRINT "ANTIGUEDAD:",A
```

Recordemos que tras suministrar un dato debemos pulsar INTRO para que el ordenador pase a solicitarnos el dato siguiente. Después de introducidos los datos se borra la pantalla (línea 60) y a continuación se imprimen de forma ordenada a modo de tabla (líneas 70 a 120). El PRINT de la línea 80 deja una línea vacía.

En la figura 1 podemos ver el resultado final de una posible ejecución de este programa.



Esta es una posible ejecución del programa 2.

Por otra parte, y como ya hemos dicho, podemos poner varias variables en un mismo INPUT, lo que nos permite abreviar los programas. Para comprobar esto podemos sustituir las líneas 20, 30, 40 y 50 del programa 2 por una sola línea:

20 INPUT "MARCA, MATRICULA, COLOR Y AÑOS DEL VEHICULO"; V$,M$,C$,A

Sin embargo, el modo de introducir los datos cuando se utiliza este formato puede variar un poco de unos ordenadores a otros. Así, en SPECTRUM y COMMODORE debemos pulsar INTRO después de introducir cada dato, mientras que en IBM, AMSTRAD o MSX se teclean todos los datos separados por comas y se pulsa INTRO una sola vez al final.

Como resumen, debemos respetar siempre las siguientes reglas sobre INPUT:

— Un único mensaje.
— Separado por punto y coma un nombre de variable.
— Y separados por comas el resto de los nombres de las variables.

El único ordenador que permite saltarse estas normas es el SPECTRUM, que admite cualquier combinación de separadores (coma y punto y coma), así como el número de mensajes que se deseen.

En ocasiones puede suceder que deseemos interrumpir la ejecución de un programa que está detenido en un INPUT esperando un dato. El modo de hacerlo varía de unas máquinas a otras, aunque siempre hay una tecla o combinación de ellas que permite hacerlo. En la figura 2 podemos ver una tabla que nos muestra el modo de detener la ejecución del programa en los principales ordenadores.

AMSTRAD	**ESC**
COMMODORE	**RUN/STOP + RESTORE**
IBM	**CTRL + BREAK**
MSX	**CTRL + STOP**
SPECTRUM	**STOP**

Métodos para detener la ejecución en un INPUT.

Finalmente, en algunos ordenadores como IBM, AMSTRAD o MSX, podemos encontrarnos que INPUT no admite separa-

dores incluidos dentro de una cadena de caracteres. Este problema se puede resolver con un nuevo formato.

LINE INPUT <Lista de variables alfanuméricas>

Su efecto es incluir todos los caracteres que se tecleen, incluidos los separadores, como valor de una sola variable hasta que pulsemos INTRO.

En los ordenadores que no disponen de LINE INPUT se puede solucionar este problema tecleando la cadena que contenga separadores entre comillas.



Comentarios en líneas de programa: REM

En muchas ocasiones manejaremos programas elaborados por otras personas. Si antes de utilizarlos no nos han explicado su funcionamiento, tendremos que estudiar detenidamente su listado, lo cual puede resultar laborioso si es muy largo. También nuestros propios programas pueden ser difíciles de reconocer si ha pasado mucho tiempo. Por estas razones siempre es conveniente introducir entre las líneas del programa algún comentario que explique su funcionamiento.

Para cubrir esta finalidad contamos con la instrucción REM, abreviatura de la palabra inglesa REMARK (nota, comentario, observación). Detrás de una instrucción REM podemos escribir cualquier comentario que deseemos. Cuando el ordenador encuentra una instrucción REM al ejecutar un programa, ignora dicha línea, es decir, no la ejecuta. Por tanto, la instrucción REM tiene utilidad para el usuario, pero no para la máquina. De todos modos, tampoco conviene abusar de las líneas REM, ya que, aunque no son ejecutables, ocupan memoria.

Veamos un ejemplo. Vamos a desarrollar un programa que nos permita calcular el volumen de un cilindro en función del radio de la base y de la altura. Recordemos que la fórmula matemática para calcularlo es: $V = 3.1416 * R^2 * H$.



En el programa la línea 10 es un mero comentario que sirve para titular el programa, por tanto, la ejecución comienza realmente en la línea 20, borrándose la pantalla. A continuación el ordenador nos pide un valor para el radio. Una vez que se lo hemos dado nos pide otro valor para la altura. Tras darle este segundo dato calcula el volumen (línea 50) e imprime el resultado en pantalla (línea 70). La línea 60 deja una línea de la pantalla en blanco.

Por último, si quisiéramos poner un título a cada uno de los programas que hemos hecho hasta el momento no tendríamos más que teclear una línea REM con el título correspondiente y con un número de línea entre 1 y 9. Así, por ejemplo, para titular el programa 1, podríamos teclear:

```
5 REM AREA Y PERIMETRO DE UNA
CIRCUNFERENCIA
```

No importa que ésta sea la última línea que tecleamos, ya que la máquina se encarga de ordenar todas las líneas que componen un programa por números de líneas crecientes. De modo que, como las líneas de programa se numeran normalmente de 10 en 10, siempre podemos intercalar alguna línea que se nos haya olvidado, utilizando un número de línea intermedio. El ordenador se encargará de situarla en la parte del programa correspondiente.

MAQUINA 8086



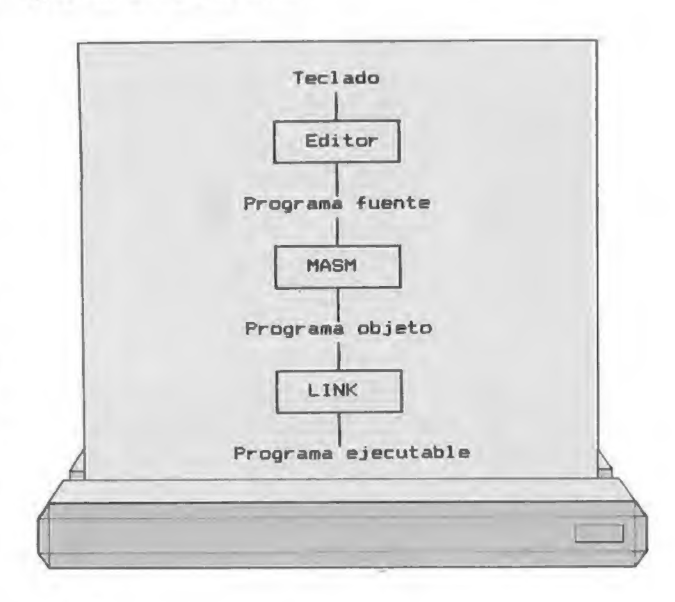
A hemos visto que mediante el mandato «A» del DEBUG se puede realizar el ensamblaje de instrucciones conforme se van escribiendo. Este procedimiento que nos ha permitido empezar a escribir y ejecutar programas desde el primer momento puede resultar instructivo, pero no se puede considerar un procedimiento general aplicable a cualquier caso.

Hay dos problemas importantes en el uso del mandato «A» del DEBUG. En primer lugar, no se pueden usar símbolos para referirse a la memoria, sino que hay que especificar las direcciones en código hexadecimal, y, en segundo lugar, no se pueden insertar nuevas instrucciones entre las que se tenían previamente escritas. Estos dos inconvenientes obligan a que las modificaciones tengan que hacerse volviendo a escribir parcial o totalmente el programa.

El método que ahora proponemos, que es el habitual, consiste en ensamblar en bloque todo el programa fuente para constituir un nuevo fichero que contiene lo que se denomina «programa objeto». El proceso completo, que se describe a continuación, consta de tres pasos:

1. Escritura o modificación del «programa fuente» utilizando cualquier editor de textos.
2. Ensamblaje del «programa fuente» y obtención del «programa objeto» mediante el programa MASM.
3. Obtención del «programa ejecutable» a partir de uno o más «programas objetos», sirviéndose del programa LINK.

El proceso puede resumirse en el siguiente esquema:



Programa fuente

Se llama «programa fuente» al fichero que contiene el texto escrito por el usuario, y que, dividido en líneas denominadas sentencias, describe en lenguaje simbólico las tareas que debe realizar el microprocesador.

Desde el punto de vista formal, el programa fuente es un fichero de texto como cualquier otro y podemos escribirlo utilizando cualquier editor de textos.

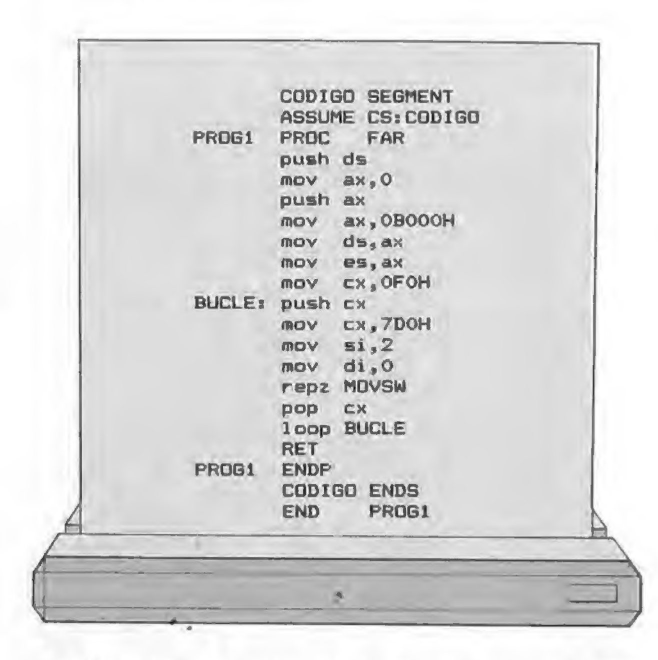
Se puede usar el EDLIN que viene incluido en el disquete del DOS. Este progra-

ma edita línea a línea y puede servir para escribir programas cortos que no vayan a ser sometidos a muchos cambios.

Para escribir programas largos o que vayan a estar sometidos a muchas modificaciones el EDLIN puede resultar demasiado tedioso. En su lugar, recomendamos el uso del «Personal Editor» o cualquier otro editor de pantalla completa.

Debido a la gran variedad de editores existentes, el manejo de los mismos es un tema muy extenso que, por otra parte, se sale del ámbito de esta sección y es un tema en el que no vamos a entrar.

Así que, sea cual sea el editor elegido, escribamos un fichero de texto de nombre PROG1.ASM que contenga el siguiente programa fuente:



```
        CODIGO SEGMENT
        ASSUME CS:CODIGO
PROG1   PROC   FAR
        push ds
        mov  ax,0
        push ax
        mov  ax,0B000H
        mov  ds,ax
        mov  es,ax
        mov  cx,0F0H
BUCLE:  push cx
        mov  cx,7D0H
        mov  si,2
        mov  di,0
        repz MOVSW
        pop  cx
        loop BUCLE
        RET
PROG1   ENDP
        CODIGO ENDS
        END    PROG1
```

Esta es una versión del programa SIMBOLO (propuesto anteriormente para ser escrito con el DEBUG), adaptada para ser ensamblada con el programa MASM. Los cambios que ha sido necesario introducir se han escrito con mayúsculas y se comentarán en otra ocasión.



Programa objeto

Como ya hemos dicho, el programa fuente no es directamente entendible por el microprocesador. Para que pueda ser entendido hay que someterlo a un proceso de traducción que se llama «ensamblaje».

Existen dos programas que pueden hacer esto. Uno de ellos es el programa MASM («Macro Assembler»), y el otro es una versión reducida del mismo llamada ASM («Assembler»). En la versión reducida se han eliminado algunas instrucciones que no son imprescindibles y puede funcionar en máquinas de 64 K en lugar de las 96 K que necesita el primero.

Mientras no se diga expresamente lo contrario, los programas que se pongan como ejemplo en esta sección podrán ser procesados indistintamente por cualquiera de los dos ensambladores citados.

En el proceso de ensamblaje intervienen los cuatro ficheros siguientes:

— El fichero que contiene el programa fuente. Cuyo nombre debe ser obligatoriamente especificado. Sin embargo, su extensión es opcional y se le asigna .ASM en caso de que sea omitida.

— Un fichero generado durante el ensamblaje, que contiene el programa objeto. Si no se especifica, su nombre se toma igual al del programa fuente y su extensión se toma .OBJ en caso de ser omitida.

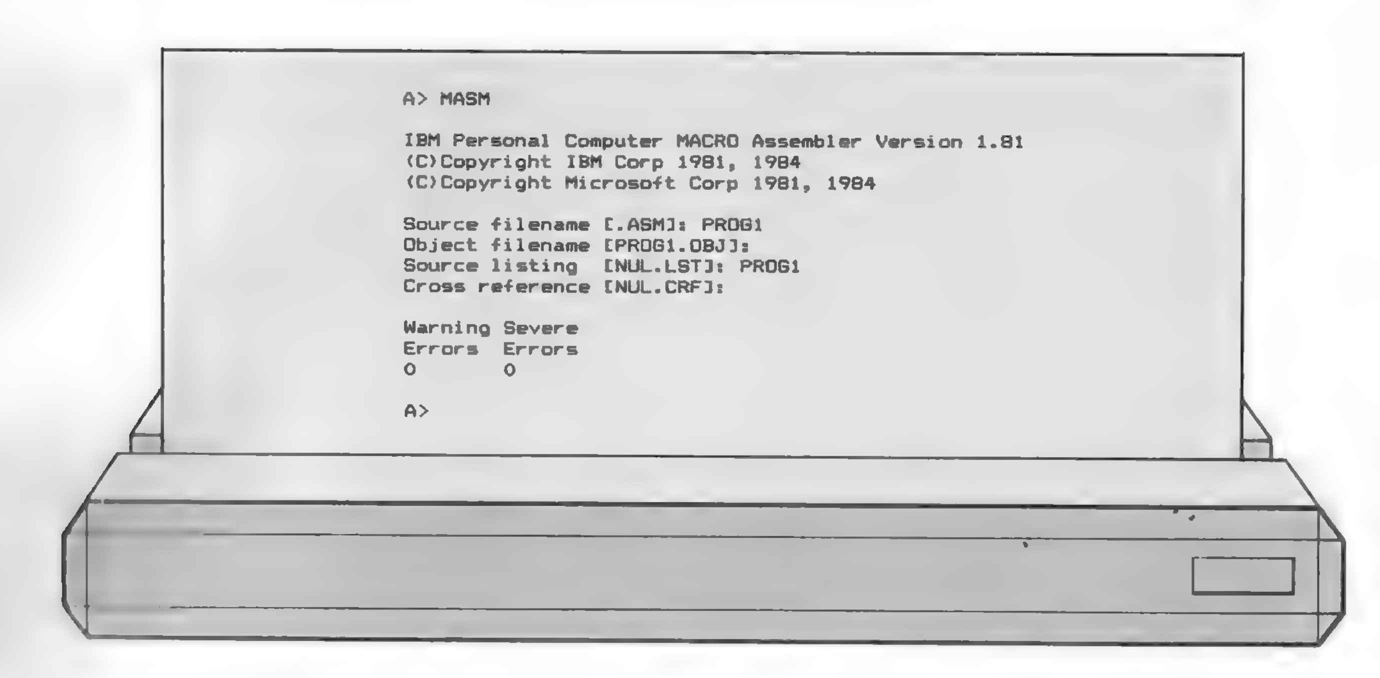
— Un fichero generado durante el ensamblaje, que contiene un listado de los programas fuente y objeto así como los posibles errores producidos. En caso de omitir su nombre y su extensión, no se genera este fichero. Si sólo se omite su extensión, se le asigna .LST.

— Un fichero generado durante el ensamblaje que contiene información sobre las referencias cruzadas. Este fichero se utiliza en casos muy excepcionales. Recomendamos omitir sistemáticamente su nombre para que no sea generado.

Hay dos formas de llamar al programa MASM.

— La primera es la forma conversacional, en ella MASM pregunta por los cuatro ficheros involucrados, indicando entre corchetes el nombre que tomará por omisión. Para los dos últimos ficheros el nombre de omisión es NUL, que significa fichero nulo, es decir, no generar ese fichero.

A continuación se expone el contenido de la pantalla cuando se ensambla el programa PROG1.ASM anteriormente listado para obtener los ficheros PROG1.OBJ y PROG1.LST usando la modalidad conversacional.



— La segunda forma de llamar al ensamblador consiste en especificar los ficheros deseados en la línea de llamada. Para ello hay que escribir MASM, seguido de un blanco, seguido de los nombres de los ficheros deseados separados por comas y terminando por punto y coma. Los nombres de los tres últimos ficheros pueden ser omitidos, en cuyo caso podrán aparecer dos o más comas seguidas.

Para especificar de esta forma los mismos ficheros que en el ejemplo anterior habría que escribir:

A> MASM PROG1,,PROG1,;

Las comas que preceden inmediatamente al punto y coma pueden omitirse.

El caso más frecuente y más sencillo de especificar es aquel en el que se quiere ensamblar el programa fuente PROG1.ASM para generar el programa objeto en el fichero PROG1.OBJ y listar directamente en la pantalla los errores encontrados. Esto se consigue especificando:

A> MASM PROG1;



Programa ejecutable

El «programa objeto» producido por el ensamblador no es todavía directamente ejecutable, sino que debe ser sometido a un último proceso llamado «montaje». Este proceso lo realiza el programa LINK, y consiste en generar un nuevo fichero que llamaremos «programa ejecutable» a partir del «programa objeto» o a partir de varios «programas objetos».

La necesidad de este paso surge cuando se realizan programas complicados. Normalmente, los programas complicados se diseñan por partes, que constituyen diferentes programas fuentes (que en muchos casos están escritos por diferentes personas).

El problema aparece cuando en un programa fuente se escribe una instrucción en la que se hace referencia a una tarea o a un dato contenido en un programa diferente. Esto recibe el nombre de «referencia externa». Cuando el ensamblador llega a una instrucción de este tipo se encuentra con el problema de que no puede traducirla, ya que no conoce la dirección de memoria de dicha «referencia externa».

¿Qué hacer en este caso? Pues lo único posible: dejar la instrucción sin traducir y poner una marca al principio del programa objeto avisando de esta circunstancia.

Cuando se han ensamblado todos los programas fuente y se procede finalmente a reunirlos en un único programa ejecutable, el LINK adjudica una posición en memoria a cada uno de los objetos. Una vez hecho esto, el LINK conoce las direcciones de todas las referencias externas y puede ya abordar la tarea de completar la traducción de las instrucciones que no había podido realizar el ensamblador.

Evidentemente, para los programas

sencillos, constituidos por un solo programa fuente, el montaje podría haberse evitado, pero no se ha hecho así. Y es ésta una pequeña incomodidad que hay que sufrir como pago a tener una herramienta mucho más general.

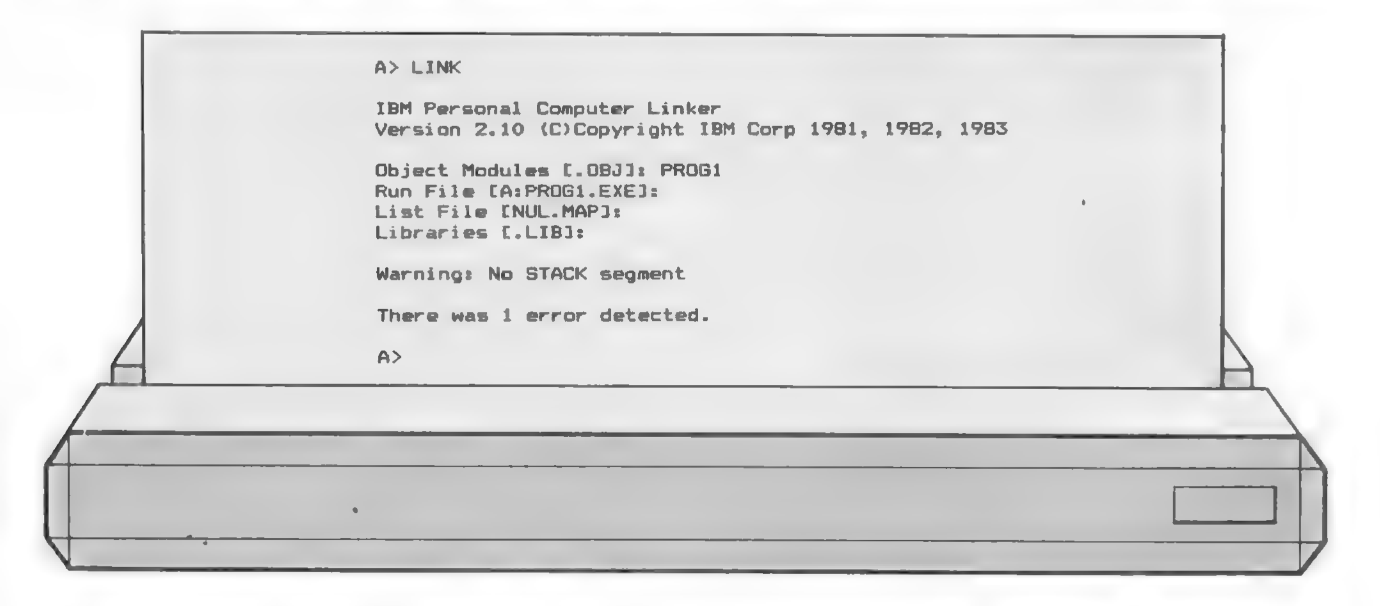
En el proceso de montaje con el LINK deben especificarse cuatro tipos de ficheros:

1. El fichero o los ficheros de los programas objetos, cuyos nombres deben formar una lista separados por blancos o por signos más (+), y cuyas extensiones pueden omitirse en caso de ser .OBJ.
2. El fichero que contendrá el programa ejecutable. Si no se especifica, su nombre se toma igual al del primer programa objeto y su extensión se toma .EXE en caso de ser omitida.
3. Un fichero generado durante el montaje, que contiene un listado con las posiciones de memoria que le han correspondido a los diferentes programas objeto, así como los posibles errores producidos. En caso de omitir su nombre y su extensión este fichero no es generado. Si sólo se omite su extensión, se le asigna .MAP.
4. El fichero o ficheros que constituyen las bibliotecas, las cuales se utilizan normalmente en el proceso de objetos producidos por los distintos compiladores. En esta sección vamos a prescindir de ellas.

El programa LINK puede utilizarse de tres formas:

— La primera forma consiste en escribir el nombre LINK y limitarse a responder los nombres de los ficheros pedidos. A continuación se expone un ejemplo de cómo obtener el programa ejecutable PROG 1.EXE a partir del programa objeto PROG1.OBJ anteriormente ensamblado.

— La segunda forma consiste en especificar los ficheros deseados en la línea de llamada. Para ello hay que escribir LINK, seguido de un blanco, seguido de los nombres de los ficheros deseados, separados por comas, y terminando por punto y coma. Los nombres de los tres últimos ficheros pueden ser omitidos, en cuyo caso podrán aparecer dos o más comas seguidas.



Para especificar de esta forma el mismo proceso que en el ejemplo anterior habría que escribir:

A > LINK PROG1,,,;

o bien, omitiendo las comas finales:

A > LINK PROG1;

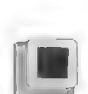
— La tercera forma, que es mediante el fichero de respuestas automáticas, puede ignorarse, ya que no se necesitará en esta sección.

Con todo esto hemos pretendido dar una visión general aunque no exhaustiva de las distintas formas en que pueden invocarse los programas MASM y LINK. Se ha dejado aparte el tema de los parámetros de estos programas, porque su utilización sólo es necesaria en casos que caen fuera del propósito de esta sección.



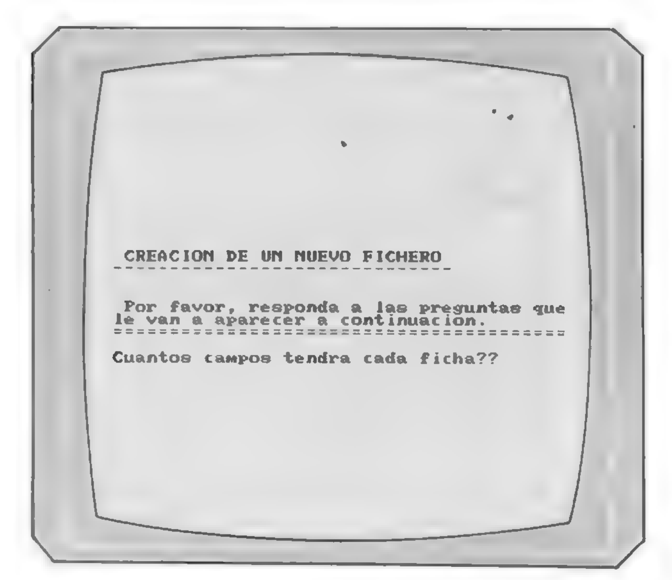
PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS



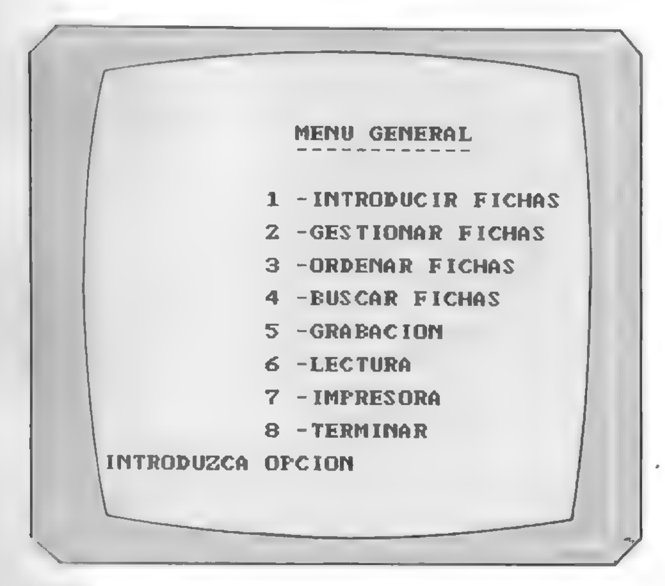
Programa: Gestor de ficheros

EL programa que aparece a continuación nos va a permitir la realización y definición de cualquier tipo de fichero que nosotros deseemos. Podremos introducir todos los datos que queramos sobre cualquier materia, ya que somos nosotros mismos los que definiremos la longitud de cada registro, de cada campo de cada registro, el nombre de cada registro...

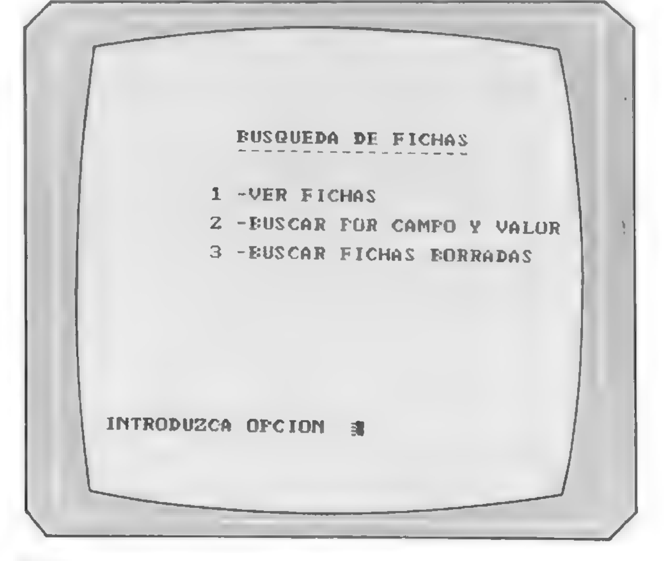



El programa permite definir los ficheros que queramos y como queramos.

El programa también está preparado para definir la impresión por impresora. Nos permite buscar fichas, borrar fichas, modificarlas...




Menú general del programa «Gestor de ficheros».




Menú de búsqueda de fichas.

Poco más hay que decir sobre el funcionamiento del programa, ya que éste es autoexplicativo.

```
100 REM **********************
110 REM * GESTOR DE FICHEROS *
120 REM **********************
130 REM
140 REM *****************************
150 REM * Por Fco. Morales Guerrero *
160 REM ******************************
170 REM * (c) Ed. Siglo Cultural,1987 *
180 REM ******************************
190 REM
200 REM *** INICIALIZACION ***
210 REM
220 DIM F$(1000),A$(10),N(20),N$(20,4)
230 LET TT=0
240 CLS
250 LET A$="MENU INICIAL"
260 LET N=2
270 LET A$(1)="DEFINIR FICHERO NUEVO"
280 LET A$(2)="ENTRAR EN EL MENU GENERAL"
290 LET M$="2"
300 LET W$="1"
310 GOSUB 5960
320 IF A$="1" THEN GOSUB 7250:GOTO 240
330 CLS
340 LET A$="MENU GENERAL"
350 LET N=8
360 LET A$(1)="INTRODUCIR FICHAS"
370 LET A$(2)="GESTIONAR FICHAS"
380 LET A$(3)="ORDENAR FICHAS"
390 LET A$(4)="BUSCAR FICHAS"
400 LET A$(5)="GRABACION"
410 LET A$(6)="LECTURA"
420 LET A$(7)="IMPRESORA"
430 LET A$(8)="TERMINAR"
440 LET M$="8"
450 GOSUB 5960
460 LET A=VAL(A$)
470 ON A GOSUB 490,650,1050,1150,1450,1990,2550,3200
480 GOTO 330
490 REM
500 REM *** INTRODUCIR FICHAS ***
510 REM
520 CLS
530 PRINT " INTRODUCIR FICHAS"
540 PRINT "=================="
550 PRINT "Fichas en memoria = ";TT
560 LET NF=TT+1:LET F$(NF)=CHR$(254)
570 GOSUB 8010
580 LET TT=TT+1
590 LOCATE 20,1
600 PRINT "INTRODUCIR OTRA FICHA (S/N)"
610 LET A$=INKEY$:IF A$="" THEN GOTO 610
620 IF A$="S" OR A$="s" THEN GOTO 490
630 IF A$="N" OR A$="n" THEN RETURN
640 GOTO 610
650 REM
660 REM *** GESTIONAR FICHAS ***
670 REM
680 LET A$="GESTION DE FICHAS"
690 LET N=4
700 LET A$(1)="BORRAR FICHA"
710 LET A$(2)="LIMPIEZA DEL FICHERO"
720 LET A$(3)="MODIFICAR FICHA"
```

```
730 LET A$(4)="VER UNA FICHA"
740 GOSUB 5960
750 CLS
760 ON VAL(A$) GOTO 770,850,890,980
770 PRINT "BORRAR FICHA"
780 PRINT "============"
790 PRINT
800 IF N(1)=0 THEN PRINT "NO HAY FICHAS":GOSUB 5440:RETURN
810 INPUT "QUE No. DE FICHA ";NF
820 IF NF<1 OR NF>77 THEN GOTO 810
830 LET F$(NF)=CHR$(253)+F$(NF)
840 RETURN
850 PRINT "LIMPIAR FICHERO"
860 PRINT "==============="
870 GOSUB 5680
880 RETURN
890 PRINT "MODIFICAR FICHA"
900 PRINT "==============="
910 PRINT
920 INPUT "QUE FICHA MODIFICAS ";NF
930 IF NF<1 OR NF>TT THEN PRINT "ESA FICHA NO EXISTE":GOSUB 5440:RETURN
940 GOSUB 8290
950 CLS
960 GOSUB 8010
970 RETURN
980 PRINT "VER UNA FICHA"
990 PRINT "============="
1000 PRINT
1010 INPUT "QUE FICHA QUIERES VER ";NF
1020 IF NF<1 OR NF>TT THEN PRINT "ESA FICHA NO EXISTE":GOSUB 5440:RETURN
1030 GOSUB 8290
1040 RETURN
1050 REM
1060 REM *** ORDENAR FICHAS ***
1070 REM
1080 CLS
1090 PRINT " ORDENAR FICHAS"
1100 PRINT "-----------------"
1110 PRINT:PRINT:PRINT
1120 INPUT "ORDENAMOS POR EL CAMPO No. :";NC
1130 GOSUB 6890
1140 RETURN
1150 REM
1160 REM *** BUSCAR FICHAS ***
1170 REM
1180 PRINT "BUSQUEDA DE FICHAS"
1190 PRINT "=================="
1200 PRINT:PRINT
1210 IF N(1)=0 THEN PRINT "NO HAY FICHAS":GOSUB 5440:RETURN
1220 LET A$="BUSQUEDA DE FICHAS"
1230 LET N=3
1240 LET A$(1)="VER FICHAS"
1250 LET A$(2)="BUSCAR POR CAMPO Y VALOR"
1260 LET A$(3)="BUSCAR FICHAS BORRADAS"
1270 LET W$="1"
1280 LET M$="3"
1290 GOSUB 5960
1300 CLS
1310 PRINT "BUSQUEDA DE FICHAS"
1320 PRINT "=================="
1330 PRINT:PRINT:LET SW=VAL(A$)-1
1340 ON VAL(A$) GOTO 1350,1390,1430
1350 INPUT "INTRODUCE PRIMERA FICHA ";N1
1360 PRINT
1370 INPUT "INTRODUCE ULTIMA FICHA  ";N2
1380 GOTO 1430
1390 INPUT "QUE VALOR HAY QUE BUSCAR ";A$
1400 PRINT
```

```
1410 INPUT "EN QUE CAMPO ";NC
1420 GOTO 1430
1430 GOSUB 6230
1440 RETURN
1450 REM
1460 REM *** GRABACION ***
1470 REM
1480 LET A$="GRABACION DEL FICHERO"
1490 LET N=2
1500 LET A$(1)="GRABACION EN DISCO"
1510 LET A$(2)="GRABACION EN CINTA"
1520 LET W$="1"
1530 LET M$="2"
1540 GOSUB 5960
1550 CLS
1560 PRINT "GRABACION DEL FICHERO"
1570 PRINT "====================="
1580 PRINT:PRINT
1590 INPUT "NOMBRE DEL FICHERO ";B$
1600 IF LEN(B$)>12 THEN GOTO 1590
1610 PRINT:PRINT
1620 PRINT "GRABANDO ";B$
1630 IF A$="2" THEN GOTO 1810
1640 OPEN B$ FOR OUTPUT AS #1
1650 PRINT #1,N(1)
1660 FOR Z=1 TO N(1)
1670    PRINT #1,N(Z+1)
1680 NEXT Z:FOR Z=1 TO N(1)
1690    PRINT #1,M$(Z)
1700 NEXT Z:PRINT #1,TT
1710 FOR Z=1 TO TT
1720    PRINT #1,F$(Z)
1730 NEXT Z
1740 FOR Z=1 TO 20
1750    FOR X=1 TO 4
1760       PRINT #1,N$(Z,X)
1770    NEXT X
1780 NEXT Z
1790 CLOSE #1
1800 RETURN
1810 OPEN "CAS:"+B$ FOR OUTPUT AS #1
1820 PRINT #1,N(1)
1830 FOR Z=1 TO N(1)
1840    PRINT #1,N(Z+1)
1850 NEXT Z
1860 FOR Z=1 TO N(1)
1870    PRINT #1,M$(Z)
1880 NEXT Z
1890 FOR Z=1 TO TT
1900    PRINT #1,F$(Z)
1910 NEXT Z
1920 FOR Z=1 TO 20
1930    FOR X=1 TO 4
1940       PRINT #1,N$(Z,X)
1950    NEXT X
1960 NEXT Z
1970 CLOSE #1
1980 RETURN
1990 REM
2000 REM *** LECTURA DEL FICHERO ***
2010 REM
2020 LET A$="LECTURA DE UN FICHERO"
2030 LET N=2
2040 LET A$(1)="LECTURA DESDE DISCO"
2050 LET A$(2)="LECTURA DESDE CASETE"
2060 LET W$="1"
2070 LET M$="2"
2080 GOSUB 5960
```

```
2090 CLS
2100 PRINT "GRABACION DEL FICHERO"
2110 PRINT "====================="
2120 PRINT:PRINT
2130 INPUT "NOMBRE DEL FICHERO ";B$
2140 IF LEN(B$)>12 THEN GOTO 2130
2150 PRINT "LEYENDO ... ";B$
2160 IF A$="2" THEN GOTO 2360
2170 OPEN B$ FOR INPUT AS #1
2180 INPUT #1,N(1)
2190 FOR Z=1 TO N(1)
2200    INPUT #1,N(Z+1)
2210 NEXT Z
2220 FOR Z=1 TO N(1)
2230    INPUT #1,M$(Z)
2240 NEXT Z
2250 INPUT #1,TT
2260 FOR Z=1 TO TT
2270    INPUT #1,F$(Z)
2280 NEXT Z
2290 FOR Z=1 TO 20
2300    FOR X=1 TO 4
2310       INPUT #1,N$(Z,X)
2320    NEXT X
2330 NEXT Z
2340 CLOSE #1
2350 RETURN
2360 OPEN "CAS"+B$ FOR INPUT AS #1
2370 INPUT #1,N(1)
2380 FOR Z=1 TO N(1)
2390    INPUT #1,N(Z+1)
2400 NEXT Z
2410 FOR Z=1 TO N(1)
2420    INPUT #1,M$(Z)
2430 NEXT Z
2440 INPUT #1,TT
2450 FOR Z=1 TO TT
2460    INPUT #1,F$(Z)
2470 NEXT Z
2480 FOR Z=1 TO 20
2490    FOR X=1 TO 4
2500       INPUT #1,N$(Z,X)
2510    NEXT X
2520 NEXT Z
2530 CLOSE #1
2540 RETURN
2550 [redacted]
2560 REM *** IMPRESORA ***
2570 REM
2580 LET A$="MENU DE IMPRESION"
2590 LET N=2
2600 LET A$(1)="DEFINIR IMPRESION"
2610 LET A$(2)="IMPRIMIR"
2620 LET W$="1"
2630 LET M$="2"
2640 GOSUB 5960
2650 IF A$="1" THEN GOSUB 3390:RETURN
2660 CLS
2670 PRINT "IMPRESION DE LAS FICHAS"
2680 PRINT "======================="
2690 PRINT:PRINT
2700 IF N$(1,1)="" THEN PRINT "IMPRESION NO DEFINIDA":GOSUB 5440:RETURN
2710 IF N(1)=0 THEN PRINT "NO HAY FICHAS":GOSUB 5440:RETURN
2720 PRINT "PREPARA LA IMPRESORA Y ..."
2730 GOSUB 5440
2740 LOCATE 5,1
2750 PRINT "IMPRIMIREMOS LAS FICHAS ";
2760 FOR Z=1 TO 20
```

```
2770    IF N$(Z,1)<>"" THEN PRINT N$(Z,1);",";:GOTO 2790
2780    LET Z=21:LET N=Z
2790 NEXT Z
2800 PRINT:PRINT
2810 PRINT "EN EL ORDEN ";
2820 FOR Z=1 TO N
2830    IF N$(Z,2)<>"" THEN PRINT N$(Z,2);",";:GOTO 2850
2840    LET Z=21
2850 NEXT Z
2860 PRINT:PRINT
2870 PRINT "TABULADAS EN LAS POSICIONES ";
2880 FOR Z=1 TO N
2890    IF N$(Z,3)<>"" THEN PRINT N$(Z,3);",";:GOTO 2910
2900    LET Z=21
2910 NEXT Z
2920 PRINT:PRINT
2930 PRINT "ENTRE FICHA Y FICHA ";
2940 IF N$(1,4)="-1" THEN PRINT "CAMBIARE DE PAGINA":GOTO 2960
2950 PRINT "IMPRIMIRE ";N$(1,4);" LINEAS EN BLANCO"
2960 GOSUB 3150
2970 LET CC=0
2980 FOR Z=1 TO TT:LET CR=1
2990    FOR R=1 TO 20:LET CD=0
3000       IF N$(R,2)="" THEN LET R=20:GOTO 3090
3010       LET CC=VAL(N$(R,2))
3020       FOR V=1 TO LEN(F$(Z))
3030          IF MID$(F$(Z),V,1)=CHR$(254) THEN LET CD=CD+1:IF CD=CC THEN LET Z1
=V+1:LET V=LEN(F$(Z))
3040       NEXT V:LPRINT TAB(VAL(N$(CR,3)));:LET CR=CR+1
3050       FOR V=Z1 TO LEN(F$(Z))
3060          IF MID$(F$(Z),V,1)=CHR$(254) THEN LET V=LEN(F$(Z)):GOTO 3080
3070          LPRINT MID$(F$(Z),V,1);
3080       NEXT V
3090    NEXT R
3100    IF N$(1,4)="-1" THEN LPRINT CHR$(12):GOSUB 3150:GOTO 3120
3110    FOR R=1 TO VAL(N$(1,4)):LPRINT:NEXT R
3120 NEXT Z
3130 LPRINT "----------------------------------------------------------------
------------"
3140 RETURN
3150 FOR S=1 TO 20
3160    IF N$(S,2)="" THEN GOTO 3180
3170    LPRINT TAB(VAL(N$(S,3)));M$(VAL(N$(S,2)));
3180 NEXT S:LPRINT "-----------------------------------------------------
--------------------"
3190 RETURN
3200 REM
3210 REM *** FIN DE LA SESION ***
3220 REM
3230 CLS
3240 PRINT "TERMINAR LA SESION"
3250 PRINT "=================="
3260 PRINT:PRINT:PRINT
3270 PRINT "ESTA SEGURO (S/N)"
3280 LET A$=INKEY$
3290 IF A$="" THEN GOTO 3280
3300 IF A$="S" OR A$="s" THEN GOTO 3330
3310 IF A$="N" OR A$="n" THEN RETURN
3320 GOTO 3280
3330 CLS
3340 PRINT "SESION TERMINADA. ADIOS"
3350 FOR I=1 TO 10 .
3360    PRINT
3370 NEXT I
3380 END
3390 REM
3400 REM ***************************
3410 REM * DEFINICION DE IMPRESION *
3420 REM ***************************
```

```
3430 REM
3440 CLS
3450 PRINT "DEFINICION DE LA IMPRESION"
3460 PRINT "=========================="
3470 LOCATE 5,1
3480 PRINT "( CUANTOS CAMPOS QUIERE QUE IMPRIMA ?"
3490 LOCATE 12,1
3500 LINE INPUT "==> ";A$
3510 IF VAL(A$)=0 THEN RETURN
3520 LET N=VAL(A$)
3530 LOCATE 5,1
3540 PRINT "( QUE CAMPOS DESEA QUE IMPRIMA ?      "
3550 LOCATE 9,1
3560 PRINT "CAMPOS A IMPRIMIR  = "
3570 FOR Z=1 TO N
3580    LOCATE 12,1
3590    PRINT SPACE$(10)
3600    LOCATE 12,1
3610    LINE INPUT "==> ";N$(Z,1)
3620    IF VAL(N$(Z,1))=0 THEN GOTO 3580
3630    LOCATE 9,22
3640    FOR X=1 TO Z
3650       PRINT N$(X.1);",";
3660    NEXT X
3670 NEXT Z
3680 PRINT CHR$(29);" "
3690 LOCATE 5,1
3700 PRINT "( EN QUE ORDEN LOS IMPRIMO ?          "
3710 PRINT "INTRODUZCA EL NUMERO DEL CAMPO EN EL "
3720 PRINT "ORDEN EN QUE QUIERE QUE SE IMPRIMA.  "
3730 LOCATE 10,1
3740 PRINT "ORDEN DE IMPRESION    "
3750 FOR Z=1 TO N
3760    LOCATE 12,1
3770    PRINT SPACE$(10)
3780    LOCATE 12,1
3790    LINE INPUT "==> ";N$(Z,2)
3800    IF VAL(N$(Z,2))=0 THEN GOTO 3760
3810    LOCATE 10,22
3820    FOR X=1 TO Z
3830       PRINT N$(X,2);",";
3840    NEXT X
3850 NEXT Z
3860 PRINT CHR$(29);" "
3870 LOCATE 12,1
3880 PRINT "          ":REM 10 ESPACIOS EN BLANCO
3890 LET NN=0
3900 FOR Z=1 TO N
3910    FOR X=1 TO N
3920       IF N$(Z,1)=N$(X,2) THEN LET NN=NN+1:LET X=N+1
3930    NEXT X
3940 NEXT Z
3950 IF NN=N THEN GOTO 4150
3960 CLS
3970 PRINT "CAMPOS A IMPRIMIR =  ";
3980 FOR Z=1 TO N
3990    PRINT N$(Z,1);",";
4000 NEXT Z
4010 PRINT CHR$(29);" "
4020 PRINT
4030 PRINT
4040 PRINT "ORDEN DE IMPRESION = ";
4050 FOR Z=1 TO N
4060    PRINT N$(Z,2);",";
4070 NEXT Z
4080 PRINT CHR$(29);" "
4090 LOCATE 12,1
4100 PRINT "NO COINCIDEN LOS CAMPOS A IMPRIMIR"
```

```
4110 PRINT "CON EL ORDEN DE IMPRESION"
4120 LET X=1:LET Y=20
4130 GOSUB 5440
4140 GOTO 3390
4150 LOCATE 15,1
4160 PRINT "( ESTA DE ACUERDO ? (S/N)"
4170 LET A$=INKEY$
4180 IF A$="S" OR A$="s" THEN GOTO 4220
4190 IF A$<>"N" AND A$<>"n" THEN GOTO 4170
4200 ERASE N$
4210 GOTO 3390
4220 FOR Z=5 TO 20
4230    LOCATE Z,1
4240    PRINT SPACE$(40)
4250 NEXT Z
4260 LOCATE 5,1
4270 PRINT "INTRODUZCA AHORA EN QUE COLUMNA"
4280 PRINT "DE LA IMPRESORA SE HA DE IMPRI-"
4290 PRINT "MIR CADA CAMPO."
4300 FOR Z=1 TO N
4310    LOCATE 12,1
4320    PRINT "CAMPO No. ";N$(Z,1);" ";
4330    LINE INPUT "==> ";N$(Z,3)
4340    IF VAL(N$(Z,3))=0 THEN GOTO 4310
4350 NEXT Z
4360 FOR Z=5 TO 17
4370    LOCATE Z,1
4380    PRINT SPACE$(40)
4390 NEXT Z
4400 LOCATE 5,1
4410 FOR Z=1 TO N
4420    PRINT "EL CAMPO ";N$(Z,1);" SE TABULARA EN LA COLUMNA ";N$(Z,3)
4430 NEXT Z
4440 LOCATE 20,1
4450 PRINT "( ESTA DE ACUERDO ? (S/N)"
4460 LET A$=INKEY$
4470 IF A$="S" OR A$="s" THEN GOTO 4500
4480 IF A$<>"N" AND A$<>"n" THEN GOTO 4460
4490 GOTO 4220
4500 FOR Z=5 TO 20
4510    LOCATE Z,1
4520    PRINT SPACE$(40)
4530 NEXT Z
4540 LOCATE 5,1
4550 PRINT "( CUANTAS LINEAS DEJO ENTRE FICHA"
4560 PRINT "Y FICHA? PULSA -1 PARA CAMBIAR DE"
4570 PRINT "PAGINA."
4580 LOCATE 12,1
4590 LINE INPUT "==> ";N$(1,4)
4600 IF VAL(N$(1,4))=0 THEN GOTO 4580
4610 CLS
4620 PRINT "OPERACION COMPLETADA"
4630 PRINT "===================="
4640 LET X=1:LET Y=20
4650 GOSUB 5440
4660 CLS
4670 RETURN
4680 REM
4690 REM *********************************
4700 REM * SUBRUTINA DE ENTRADA DE DATOS *
4710 REM *********************************
4720 REM
4730 LET D$="" : LET LO=0
4740 LOCATE Y,X
4750 FOR Z=1 TO LO
4760    PRINT ".";
4770 NEXT Z
4780 LOCATE Y,X
4790 PRINT "_"
```

```
4800 LOCATE Y,X
4810 LET A$=INKEY$
4820 IF A$="" THEN GOTO 4810
4830 IF A$=CHR$(8) AND LO>0 THEN LET LO=LO-1:LET D$=LEFT$(D$,LO) :LET X=X-1:LOCA
TE Y,X:PRINT "_.":LOCATE Y,X:GOTO 4810
4840 IF A$=" " OR A$="." OR A$="-" THEN GOTO 4870
4850 IF A$=CHR$(13) THEN GOTO 4930
4860 IF A$>M$ OR A$<W$ THEN GOTO 4810
4870 PRINT A$;"_"
4880 LET X=X+1
4890 LOCATE Y,X
4900 LET D$=D$+A$
4910 LET LO=LO+1
4920 IF LO<>LO THEN GOTO 4810
4930 LOCATE Y,X
4940 FOR Z=LO TO LO
4950    PRINT " ";
4960 NEXT Z
4970 RETURN
4980 REM
4990 REM ********************************
5000 REM * MENSAJE Y RECOGIDA DE OPCION *
5010 REM ********************************
5020 REM
5030 LET T$="INTRODUZCA OPCION"
5040 LET L=17
5050 LOCATE Y,X
5060 PRINT T$;
5070    LOCATE Y,X+19
5080    PRINT CHR$(177);
5090    LET A$=INKEY$
5100    IF A$<>"" AND (A$<W$ OR A$>M$) THEN GOSUB 5250
5110    IF A$<>"" THEN GOSUB 5360:GOTO 5220
5120    FOR Z=1 TO 100
5130    NEXT Z
5140    LOCATE Y,X+19
5150    PRINT " ";
5160    LET A$=INKEY$
5170    IF A$<>"" AND (A$<W$ OR A$>M$) THEN GOSUB 5250
5180    IF A$<>"" THEN GOSUB 5360:GOTO 5220
5190    FOR Z=1 TO 100
5200    NEXT Z
5210 GOTO 5070
5220 LOCATE Y,X
5230 PRINT SPACE$(L+3);
5240 RETURN
5250 REM
5260 REM *** ERROR ***
5270 REM
5280 LOCATE Y,X+19
5290 PRINT "ERROR ..."
5300 FOR Z=1 TO 500
5310 NEXT Z
5320 LOCATE Y,X+19
5330 PRINT "         ":REM 9 ESPACIOS
5340 LET A$=""
5350 RETURN
5360 REM
5370 REM *** IMPRESION DE LA OPCION ELEGIDA ***
5380 REM
5390 LOCATE Y,X+19
5400 PRINT A$;
5410 FOR Z=1 TO 500
5420 NEXT Z
5430 RETURN
5440 REM
5450 REM ******************
5460 REM * PULSA UNA TECLA *
5470 REM ******************
```

```
5480 REM
5490 LET A$="PULSA UNA TECLA"
5500 LET L=15
5510    BEEP
5520    FOR Z=1 TO 6
5530       LOCATE Y,X
5540       PRINT A$;
5550       IF INKEY$<>"" THEN GOTO 5650
5560       FOR K=1 TO 100
5570       NEXT K
5580       LOCATE Y,X
5590       PRINT SPACE$(L);
5600       IF INKEY$<>"" THEN GOTO 5650
5610       FOR K=1 TO 100
5620       NEXT K
5630    NEXT Z
5640 GOTO 5510
5650 LOCATE Y,X
5660 PRINT SPACE$(L);
5670 RETURN
5680 REM
5690 REM ***********************************
5700 REM * SUBRUTINA DE BORRADO REAL DE FICHAS *
5710 REM ***********************************
5720 REM
5730 LET N=0
5740 FOR Z=1 TO TT
5750    IF LEFT$(F$(Z),1)<>CHR$(253) THEN GOTO 5810
5760    LET F$(Z)=""
5770    LET N=N+1
5780    FOR X=Z TO TT-1
5790       LET F$(X)=F$(X+1)
5800    NEXT X
5810 NEXT Z
5820 FOR Z=TT-N+1 TO TT
5830    LET F$(Z)=""
5840 NEXT Z
5850 LET TT=TT-N
5860 RETURN
5870 PRINT "E R R O R"
5880 PRINT "---------"
5890 PRINT
5900 PRINT "Lo siento pero no he encontrado el campo No. ";NC
5910 PRINT
5920 PRINT "Estas seguro de que las fichas de este fichero tienen";NC;"campos?"
5930 X=1:Y=20:GOSUB 5440
5940 CLS
5950 RETURN
5960 REM
5970 REM ******************************
5980 REM * SUBRUTINA DE CREACION DE MENUS *
5990 REM ******************************
6000 REM
6010 CLS
6020 LET X=INT((40-LEN(A$))/2)
6030 LOCATE 1,X
6040 PRINT A$
6050 LOCATE 2,X
6060 FOR Z=1 TO LEN(A$)
6070    PRINT "-";
6080 NEXT Z
6090 PRINT:PRINT
6100 LET X=0
6110 FOR Z=1 TO N
6120    IF X<LEN(A$(Z)) THEN LET X=LEN(A$(Z))
6130 NEXT Z
6140 LET X=INT((40-X)/2)
6150 FOR Z=1 TO N
```

```
6160    LOCATE 2*Z+3,X
6170    PRINT Z;"-";A$(Z)
6180 NEXT Z
6190 LET X=1:LET Y=21
6200 GOSUB 4980
6210 CLS
6220 RETURN
6230 REM
6240 REM *************************
6250 REM * SUBRUTINA DE BUSQUEDA *
6260 REM *************************
6270 REM
6280 CLS
6290 PRINT "B U S C A N D O"
6300 PRINT "==============="
6310 PRINT
6320 IF SW=0 THEN PRINT "Buscando las fichas desde la ";N1;" hasta la ";N2
6330 IF SW=1 THEN PRINT "Buscando las fichas que tengan ";A$;" en el campo ";NC
6340 IF SW=2 THEN PRINT "Buscando fichas borradas"
6350 PRINT
6360 PRINT "ESPERA UN MOMENTO"
6370 FOR Z=1 TO 1000
6380 NEXT Z
6390 CLS
6400 ON SW+1 GOSUB 6490,6580,6770
6410 CLS
6420 PRINT "OPERACION TERMINADA"
6430 PRINT "==================="
6440 PRINT
6450 PRINT "No se han encontrado mas fichas."
6460 LET X=1:LET Y=20
6470 GOSUB 5440
6480 RETURN
6490 REM
6500 REM *** BUSQUEDA POR No. DE FICHA ***
6510 REM
6520 FOR R=N1 TO N2
6530    LET NF=R
6540    GOSUB 8290
6550    CLS
6560 NEXT R
6570 RETURN
6580 REM
6590 REM *** BUSQUEDA POR CAMPO Y VALOR ***
6600 REM
6610 IF N(1)<>0 THEN GOTO 6690
6620 PRINT "E R R O R"
6630 PRINT "---------"
6640 PRINT
6650 PRINT "Lo siento. El campo ";NC;" no existe."
6660 GOSUB 5440
6670 CLS
6680 RETURN
6690 LET J$=A$:FOR Z=1 TO TT:LET CD=0:LET CR=0:LET Z1=0
6700    FOR X=1 TO LEN(F$(Z)):IF MID$(F$(Z),X,1)=CHR$(254) THEN LET CD=CD+1:IF C
D=NC THEN LET Z1=X+1:LET X=LEN(F$(Z))
6710    NEXT X:FOR X=Z1 TO LEN(F$(Z)):IF MID$(J$,X-Z1+1,1)=MID$(F$(Z),X,1) THEN
LET CR=CR+1:IF CR=LEN(J$) THEN LET X=LEN(F$(Z))
6720    NEXT X
6730    IF CR=LEN(J$) THEN LET VV=Z:LET NF=Z:GOSUB 8290:LET Z=VV
6740 NEXT Z
6750 RETURN
6760    CLS
6770 REM
6780 REM *** BUSQUEDA DE FICHAS BORRADAS ***
6790 REM
6800 FOR Z=1 TO TT
6810    IF LEFT$(F$(Z),1)<>CHR$(253) THEN GOTO 6870
```

```
6820    LET F$(Z)=RIGHT$(F$(Z),LEN(F$(Z))-1)
6830    LET NF=Z
6840    GOSUB 8290
6850    CLS:LET Z=NF
6860    LET F$(Z)=CHR$(253)+F$(Z)
6870 NEXT Z
6880 RETURN
6890 REM
6900 REM ***************************
6910 REM * ORDENACION ALFANUMERICA *
6920 REM ***************************
6930 REM
6940 CLS
6950 PRINT "O R D E N A R   F I C H E R O"
6960 PRINT "=============================="
6970 PRINT
6980 PRINT "Ordenando fichero por el campo No. ";NC
6990 PRINT
7000 PRINT "ESPERA UN MOMENTO"
7010 IF N(1)>=NC THEN GOTO 7120
7020 CLS
7030 PRINT "E R R O R"
7040 PRINT "---------"
7050 PRINT
7060 PRINT "Lo siento pero no he encontrado el campo No. ";NC
7070 PRINT
7080 PRINT "Estas seguro de que las fichas de este fichero tienen";NC;"campos?"
7090 X=1:Y=20:GOSUB 5440
7100 CLS
7110 RETURN
7120 FOR Z=1 TO TT:LET CC=0:LET CD=0
7130    FOR X=1 TO Z:FOR R=1 TO LEN(F$(Z)):IF MID$(F$(Z),R,1)=CHR$(254) THEN LET
 CC=CC+1:IF CC=NC THEN LET Z1=R+1:LET R=LEN(F$(Z))
7140    NEXT R:LET CC=0:FOR R=1 TO LEN(F$(X)):IF MID$(F$(X),R,1)=CHR$(254) THEN
LET CC=CC+1:IF CC=NC THEN LET Z2=R+1:LET R=LEN(F$(X))
7150    NEXT R:IF MID$(F$(Z),Z1)<MID$(F$(X),Z2) THEN LET A$=F$(Z):LET F$(Z)=F$(X
):LET F$(X)=A$
7160    NEXT X
7170 NEXT Z
7180 CLS:PRINT "O P E R A C I O N   T E R M I N A D A"
7190 PRINT "====================================="
7200 PRINT
7210 PRINT "El fichero esta ordenado por el campo No. ";NC
7220 X=1:Y=20:GOSUB 5440
7230 CLS
7240 RETURN
7250 REM
7260 REM **********************************
7270 REM * DEFINICION DE UN FICHERO NUEVO *
7280 REM **********************************
7290 REM
7300 CLS
7310 LET A$="CREACION DE UN FICHERO"
7320 LET N=2
7330 LET A$(1)="UTILIZAR UNO QUE ESTE CREADO"
7340 LET A$(2)="CREAR UNO NUEVO"
7350 GOSUB 5960
7360 IF A$="1" THEN GOTO 7640
7370 PRINT " CREACION DE UN NUEVO FICHERO"
7380 PRINT "-----------------------------"
7390 PRINT:PRINT
7400 PRINT " Por favor, responda a las preguntas quele van a aparecer a continua
cion."
7410 PRINT "======================================"
7420 LOCATE 9,1
7430 INPUT "Cuantos campos tendra cada ficha?";N(1)
7440 IF N(1)<0 OR N(1)>20 THEN GOTO 7420
7450 DIM M$(N(1)+1)
```

```
7460 PRINT
7470 FOR Z=1 TO N(1)
7480    LOCATE 16,1
7490    PRINT "                                                      "
7500    LOCATE 18,1
7510    PRINT "                                                      "
7520    LOCATE 16,1
7530    PRINT "Cuantos caracteres ocupa el campo ";Z
7540    INPUT N(Z+1)
7550    LOCATE 18,1
7560    PRINT "Como se llama este campo"
7570    INPUT M$(Z)
7580 NEXT Z
7590 CLS
7600 PRINT "OPERACION TERINADA"
7610 LET X=1:LET Y=20
7620 GOSUB 5440
7630 RETURN
7640 REM
7650 REM *** UTILIZACION DE UN FICHERO YA CREADO ***
7660 REM
7670 CLS
7680 PRINT " CREACION DE UN FICHERO"
7690 PRINT "======================="
7700 PRINT
7710 PRINT " Teclea el nombre del fichero que ya tiene una estructura definida y
 del cual laquieres copiar."
7720 PRINT:PRINT:PRINT
7730 INPUT "NOMBRE = ";N$
7740 PRINT:PRINT
7750 PRINT " Asegurate de que esta colocada la cintao disco que contiene dicho f
ichero"
7760 LET X=1:LET Y=20
7770 GOSUB 5440
7780 CLS
7790 PRINT "LEYENDO ... ";N$
7800 OPEN N$ FOR INPUT AS #1
7810 INPUT#1,T
7820 DIM M$(T+1)
7830 FOR Z=1 TO T
7840    INPUT#1,N(Z+1)
7850 NEXT Z
7860 LET N(1)=T
7870 FOR Z=1 TO T
7880    INPUT#1,M$(Z)
7890 NEXT Z
7900 CLOSE#1:CLS
7910 PRINT "CONFIGURACION LEIDA"
7920 PRINT:PRINT
7930 PRINT "Cada ficha tiene ";T;" campos"
7940 PRINT
7950 FOR Z=1 TO T
7960    PRINT "el campo ";Z;" se llama ";M$(Z)
7970 NEXT Z
7980 LET X=1:LET Y=21
7990 GOSUB 5440
8000 RETURN
8010 REM
8020 REM *************************
8030 REM * INTRODUCCION DE FICHAS *
8040 REM *************************
8050 REM
8060 PRINT
8070 PRINT "INTRUDUCIENDO LA FICHA ";NF
8080 PRINT
8090 FOR R=1 TO N(1)
8100    LOCATE 10,1
8110    PRINT M$(R);" : ";
```

```
8120    LET X=LEN(M$(R))+4
8130    LET Y=10
8140    LET M$="z"
8150    LET W$="0"
8160    LO=N(R+1):GOSUB 4680
8170    LOCATE 20,1
8180    PRINT "ESTA CORRECTO ESTE CAMPO (S/N)"
8190    LET A$=INKEY$:IF A$="" THEN GOTO 8190
8200    IF A$="N" OR A$="n" THEN GOTO 8100
8210    IF A$<>"S" AND A$<>"s" THEN GOTO 8190
8220    LET F$(NF)=F$(NF)+D$+CHR$(254)
8230    LOCATE 10,1
8240    PRINT SPACE$(80)
8250    LOCATE 20,1
8260    PRINT SPACE$(40)
8270 NEXT R
8280 RETURN
8290 REM
8300 REM *****************
8310 REM * VER UNA FICHA *
8320 REM *****************
8330 REM
8340 CLS:LET CC=1
8350 PRINT "FICHA No. ";NF
8360 PRINT:PRINT M$(CC);" ";
8370 IF LEFT$(F$(NF),1)=CHR$(253) THEN PRINT "ESTA FICHA ESTA BORRADA":GOSUB 54
0:RETURN
8380 FOR Z=2 TO LEN(F$(NF))
8390    IF MID$(F$(NF),Z,1)=CHR$(254) THEN LET CC=CC+1:PRINT:PRINT M$(CC);" ";:G
OTO 8410
8400    PRINT MID$(F$(NF),Z,1);
8410 NEXT Z
8420 GOSUB 5440
8430 RETURN
```

COMMODORE

```
240 PRINT CHR$(147)
330 PRINT CHR$(147)
520 PRINT CHR$(147)
590 POKE 214, 19:POKE 211,Ø
610 GET A$:IF A$"" THEN GOTO 610
750 PRINT CHR$(147)
950 PRINT CHR$(147)
1080 PRINT CHR$(147)
1300 PRINT CHR$(147)
1550 PRINT CHR$(147)
1640 OPEN 1,8,1,B$
1810 OPEN 1,1,1,B$
2090 PRINT CHR$(147)
2170 OPEN 1,8,0,B$
2360 OPEN 1,1,0,B$
2660 PRINT CHR$(147)
2740 POKE 214,4:POKE 211,0
2975 OPEN 1,4:CMD 1
3040 NEXT V:PRINT TAB(VAL(N$(CR,3)));:LET
CR=CR+1
3070 PRINT MID$(F$(Z) ,V,1);
3100 IF N$(1,4)="-1" THEN PRINT
CHR$(147):GOSUB 3150:GOTO 3120
3110 FOR R=1 TO VAL(N$(1,4)):PRINT:NEXT
R
3130 PRINT
```

```
"-----------------------------------------------------------"
3140 CLOSE #1:RETURN
3170 PRINT
TAB(VAL(N$(S,3)));M$(VAL(N$(S,2)));
3180 NEXT S:PRINT
"-----------------------------------------------------------"
3230 PRINT CHR$(147)
3280 GET A$
3330 PRINT CHR$(147)
3440 PRINT CHR$(147)
3470 POKE 214,4:POKE 211,0
3490 POKE 214,11:POKE 211,0
3530 POKE 214,4:POKE 211,0
3550 POKE 214,8:POKE 211,0
3580 POKE 214,11:POKE 211,0
3590 PRINT "          ":REM 10 ESPACIOS
3600 POKE 214,11:POKE 211,0
3630 POKE 214,8:POKE 211,21
3680 PRINT "<CURSOR-IZ>";" "
3690 POKE 214,4:POKE 211,0
3730 POKE 214,9:POKE 211,0
3760 POKE 214,11:POKE 211,0
3770 PRINT "          ":REM 10 ESPACIOS
3780 POKE 214,11:POKE 211,0
3810 POKE 214,9:POKE 211,21
3860 PRINT "<CURSOR-IZQ>";" "
3870 POKE 214,11:POKE 211,0
3960 PRINT CHR$(147)
4010 PRINT "<CURSOR-IZQ>";" "
4080 PRINT "<CURSOR-IZQ>";" "
4090 POKE 214,11:POKE 211,0
4150 POKE 214,14:POKE 211,0
4170 GET A$
4200 REM
4230 POKE 214,Z-1:POKE 211,0
4240 FOR Q=1 TO 40:PRINT " ";:NEXT Q
4260 POKE 214,4:POKE 211,0
4310 POKE 214,11:POKE 211,0
4370 POKE 214,Z-1:POKE 211,0
4380 FOR Q=1 TO 40:PRINT " ";:NEXT Q
4400 POKE 214,4:POKE 211,0
4440 POKE 214,19:POKE 211,0
4460 GET A$
4510 POKE 214,Z-1:POKE 211,0
4520 FOR Q=1 TO 40:PRINT " ";:NEXT Q
4540 POKE 214,4:POKE 211,0
4580 POKE 214,11:POKE 211,0
4610 PRINT CHR$(147)
4660 PRINT CHR$(147)
4740 POKE 214,Y:POKE 211,X
4780 POKE 214,Y:POKE 211,X
4800 POKE 214,Y:POKE 211,X
4810 GET A$
4830 IF A$=CHR$(20) AND LO>0 THEN LET
LO=LO-1:LET D$=LEFT$(D$,LO):LET
X=X-1:POKE 214,Y:POKE 211,X:PRINT
"-.":POKE 214,Y:POKE 211,X:GOTO 4810
4890 POKE 214,Y:POKE 211,X
4930 POKE 214,Y:POKE 211,X
5050 POKE 214,Y:POKE 211,X
5070 POKE 214,Y:POKE 211,X+19
5080 PRINT CHR$(102);
5990 GET A$
5140 POKE 214,Y:POKE 211,X+19
5160 GET A$
5220 POKE 214,Y:POKE 211,X
5230 FOR Q=1 TO L+3:PRINT " ";:NEXT Q
5280 POKE 214,Y:POKE 211,X+19
5320 POKE 214,Y:POKE 211,X+19
5390 POKE 214,Y:POKE 211,X+19
5510 REM
5530 POKE 214,Y:POKE 211,X
5550 GET B$:IF B$<>"" THEN GOTO 5650
5580 POKE 214,Y:POKE 211,X
5590 FOR Q=1 TO L:PRINT " ";:NEXT Q
5600 GET B$:IF B$<>" "THEN GOTO 5650
5650 POKE 214,Y:POKE 211,X
5660 FOR Q=1 TO L:PRINT " ";:NEXT Q
5940 PRINT CHR$(147)
6010 PRINT CHR$(147)
6030 POKE 214,1:POKE 211,X
6050 POKE 214,2:POKE 211,X
6160 POKE 214,2*Z+3:POKE 211,X
6210 PRINT CHR$(147)
6280 PRINT CHR$(147)
6390 PRINT CHR$(147)
6410 PRINT CHR$(147)
6550 PRINT CHR$(147)
6670 PRINT CHR$(147)
6760 REM
6850 PRINT CHR$(147):LET Z=NF
6940 PRINT CHR$(147)
7020 PRINT CHR$(147)
7100 PRINT CHR$(147)
7180 PRINT CHR$(147);" O P E R A C I O N
  T E R M I N A D A"
7230 PRINT CHR$(147)
7300 PRINT CHR$(147)
7420 POKE 214,8:POKE 211,0
7480 POKE 214,15:POKE 211,0
7500 POKE 214,17:POKE 211,0
7520 POKE 214,15:POKE 211,0
7550 POKE 214,17:POKE 211,0
7590 PRINT CHR$(147)
7670 PRINT CHR$(147)
7780 PRINT CHR$(147)
7800 OPEN 1,1,0,N$
7900 CLOSE #1:PRINT CHR$(147)
8100 POKE 214,9:POKE 211,0
8170 214,19:POKE 211,0
8190 GET A$:IF A$="" THEN GOTO 8190
8230 POKE 214,9:POKE 211,0
8240 FOR Q=1 TO 80:PRINT " ";:NEXT Q
8250 POKE 214,19:POKE 211,0
```

```
8260 FOR Q=1 TO 80:PRINT " ";:NEXT Q
8340 PRINT CHR$(147):LET CC=1
```

SPECTRUM:

```
 220 DIM F$(200,100):DIM A$(10,20):DIM
N(20):DIM N$(20,4,3)
 470 GOTO
490*(A=1)+650*(A=2)+1050*(A=3)+
1150*(A=4)+145*(A=5)+1990*(A=6)+
2550*(A=7)+3200*(A=8)
 590 PRINT AT 19,0;
 760 LET A=VAL(A$):GOTO
770*(A=1)+850*(A=2)+890*(A=3)+
980*(A=4)
1340 LET A=VAL(A$):GOTO
1350*(A=1)+1390*(A=2)+1430*(A=3)
1600 IF LEN(B$) >10 THEN GOTO 1590
1640 SAVE *"M";1;B$+"1" DATA N()
1650 SAVE *"M";1;B$+"2" DATA M$()
1660 SAVE *"M";1;B$+"3" DATA F$()
1670 SAVE *"M";1;B$+"4" DATA N$()
1680 REM
1690 REM
1700 REM
1710 REM
1720 REM
1730 REM
1740 REM
1750 REM
1760 REM
1770 REM
1780 REM
1790 REM
1810 SAVE B$+"1" DATA N()
1820 SAVE B$+"2" DATA M$()
1830 SAVE B$+"3" DATA F$()
1840 SAVE B$+"4" DATA N$()
1850 REM
1860 REM
1870 REM
1880 REM
1890 REM
1900 REM
1910 REM
1920 REM
1930 REM
1940 REM
1950 REM
1960 REM
1970 REM
2140 IF LEN(B$)>10 THEN GOTO 2130
2170 LOAD *"M";1;B$+"1" DATA N()
2180 LOAD *"M";1;B$+"2" DATA M$()
2190 LOAD *"M";1;B$+"3" DATA F$()
2200 LOAD *"M";1;B$+"4" DATA N$()
2210 FOR Q=1 TO 200
2220 IF F$(Q)="" THEN LET TT=Q-1:LET
Q=200
2230 NEXT Q
2240 REM
2250 REM
2260 REM
2270 REM
2280 REM
2290 REM
2300 REM
2310 REM
2320 REM
2330 REM
2340 REM
2360 LOAD *"M";1;B$+"1" DATA N()
2370 LOAD *"M";1;B$+"2" DATA M$()
2380 LOAD *"M";1;B$+"3" DATA F$()
2390 LOAD *"M";1;B$+"4" DATA N$()
2400 FOR Q=1 TO 200
2410 IF F$(Q)="" THEN LET TT=Q-1:LET
Q=200
2420 NEXT Q
2430 REM
2440 REM
2450 REM
2460 REM
2470 REM
2480 REM
2490 REM
2500 REM
2510 REM
2520 REM
2530 REM
2740 PRINT AT A,0;
3030 IF F$(Z,V)=CHR$(254) THEN LET
CD=CD+1:IF CD=CC THEN LET Z1=V+1:LET
V=LEN(F$(Z))
3060 IF F$(Z,V)=CHR$(254) THEN LET
V=LEN(F$Z)):GOTO 3080
3070 LPRINT F$(Z,V);
3470 PRINT AT 4,0;
3490 PRINT AT 11,0;
3500 INPUT LINE "==> ";A$
3530 PRINT AT 4,0;
3550 PRINT AT 8,0;
3580 PRINT AT 11,0;
3590 PRINT "          ":REM 10 ESPACIOS
3600 PRINT AT 11,0;
3610 INPUT LINE "==> ";N$(Z,1)
3630 PRINT AT 8,21;
3680 REM
3690 PRINT AT 4,0;
3730 PRINT AT 9,0;
3760 PRINT AT 11,0;
3770 PRINT "          ":REM 10 ESPACIOS
3780 PRINT AT 11,0;
3790 INPUT LINE "==>;N$(Z,2)
```

```
3810 PRINT AT 9,21;
3860 REM
3870 PRINT AT 11,0;
4010 REM
4080 REM
4090 PRINT AT 11,0;
4150 PRINT AT 14,0;
4200 REM
4230 PRINT AT Z-1,0;
4240 FOR Q=1 TO 40:PRINT " ";:NEXT Q
4260 PRINT AT 4,0;
4310 PRINT AT 11,0;
4330 INPUT LINE "==> ";N;$(Z,3)
4370 PRINT AT Z-1,0;
4380 FOR Q=1 TO 40:PRINT " ";:NEXT Q
4400 PRINT AT 4,0;
4440 PRINT AT 19,0;
4510 PRINT AT Z-1,0;
4520 FOR Q=1 TO 40:PRINT " ";:NEXT Q
4540 PRINT AT 4,0;
4580 PRINT AT 11,0;
4590 INPUT LINE "==> ";N$(1,4)
4740 PRINT AT Y,X;
4780 PRINT AT Y,X;
4800 PRINT AT Y,X;
4830 IF A$=CHR$(8) AND L0>0 THEN LET L0=L0-1:LET D$=D$ ( TO L0):LET X=X-1:PRINT AT Y,X;"-.";AT Y,X;:GOTO 4810
4890 PRINT AT Y,X;
4930 PRINT AT Y,X;
5050 PRINT AT Y,X;
5070 PRINT AT AT T,X+19;
5080 PRINT PAPER 0, " "
5140 PRINT AT Y,X+19;
5220 PRINT AT Y,X;
5230 FOR Q=1 TO L+3:PRINT " ";:NEXT Q
5280 PRINT AT Y,X+19;
5320 PRINT AT Y,X+19;
5390 PRINT AT Y,X+19;
5510 BEEP .2,15
5530 PRINT AT Y,X;
5580 PRINT AT Y,X;
5590 FOR Q=1 TO L:PRINT " ";:NEXT Q
5650 PRINT AT Y,X;
5660 FOR Q=1 TO L:PRINT " ";:NEXT Q
5750 IF F$(Z,1)<> CHR$(253) THEN GOTO 5810
6030 PRINT AT 0,X;
6050 PRINT AT 1,X;
6160 PRINT AT 2*Z+2,X;
6400 GOTO 6490*(SW=0)+6580*(SW=1)+6770*(SW=2)
6700 FOR X=1 TO LEN(F$(Z)):IF F$(Z,X)=CHR$(254) THEN LET CD=CD+1:IF CD=NC THEN LET Z1=X+1:LET X=LEN(F$(Z))
6710 NEXT X:FOR Z=Z1 TO LEN(F$(Z)):IF J$(X-Z1+1)=F$(Z,X) THEN LET CR=CR+1:IF CR=LEN(J$) THEN LET X=LEN(F$(Z))
6810 IF F$(Z,1)<>CHR$(253) THEN GOTO 6870
6820 LET F$(Z)=F$(Z, 2 TO)
7130 FOR Z=1 TO Z:FOR R=1 TO LEN(F$(Z)):IF F$(Z,R)=CHR$(254) THEN LET CC=CC+1:IF CC=NC THEN LET Z1=R+1:LET R=LEN(F$(Z))
7140 NEXT R:LET CC=0:FOR R=1 TO LEN(F$(X)):IF F$(X,R)=C CHR$(254) THEN LET CC=CC+1:IF CC=NC THEN LET Z2R+1:LET R=LEN(F$(X))
7150 NEXT R:IF F$(Z,Z1 TO)<F$(X,Z2 TO) THEN LET A$=F$(Z):LET F$(Z)=F$(X):LET F$(X)=A$
7420 PRINT AT 8,0;
7450 DIM M$(N(1)+1,20)
7480 PRINT AT 15,0;
7500 PRINT AT 17,0;
7520 PRINT AT 15,0;
7550 PRINT AT 17,0;
7800 LOAD B$ DATA N()
7810 LOAD B$ DATA M$()
7820 REM
7830 REM
7840 REM
7850 REM
7860 REM
7870 REM
7880 REM
7890 REM
7900 CLS
8100 PRINT AT 9,0;
8170 PRINT AT 19,0;
8230 PRINT AT 9,0;
8240 FOR Q=1 TO 80:PRINT " ";:NEXT Q
8250 PRINT AT 19,0;
8260 FOR Q=1 TO 80:PRINT " ";:NEXT Q
8370 IF F$(NF,1)=CHR$(253) THEN PRINT "ESTA FICHA ESTA BORRADA":GOSUB 5440:RETURN
8390 IF F$(Z,1)=CHR$(254) THEN LET CC=CC+1:PRINT:PRINT M$(CC); " ";:GOSUB 8410
8400 PRINT F$(NF,Z);
```

MSX Y AMSTRAD:

```
 590 LOCATE 1,20
2740 LOCATE 1,5
3470 LOCATE 1,5
3490 LOCATE 1,9
3580 LOCATE 1,12
3600 LOCATE 1,12
3630 LOCATE 22,9
3690 LOCATE 1,5
3730 LOCATE 1,10
3760 LOCATE 1,12
```

```
3780 LOCATE 1,12
3810 LOCATE 22,10
3870 LOCATE 1,12
4090 LOCATE 1,12
4150 LOCATE 1,15
4230 LOCATE 1,7
4260 LOCATE 1,5
4310 LOCATE 1,12
4370 LOCATE 1,Z
4400 LOCATE 1,5
4440 LOCATE 1,20
4510 LOCATE 1,Z
4540 LOCATE 1,5
4580 LOCATE 1,12
4740 LOCATE X,Y
4780 LOCATE X,Y
4800 LOCATE X,Y
4830 IF A$=CHR$(8) AND L0>0 THEN LET
L0=L0-1:LET D$=LEFT$(D$,L0):LET
X=X-1:LOCATE Y,Y:PRINT "-.":LOCATE
X,Y:GOTO 4810
4890 LOCATE X,Y
4930 LOCATE X,Y
5050 LOCATE X,Y
5070 LOCATE X+19,Y
5140 LOCATE X=19,Y
5220 LOCATE X,Y
5280 LOCATE X+19,Y
5320 LOCATE X+19,Y
5390 LOCATE X+19,Y
5530 LOCATE X,Y
5580 LOCATE X,Y
5650 LOCATE X,Y
6030 LOCATE X,1
6050 LOCATE X,2
6160 LOCATE X,2*Z+3
7420 LOCATE 1,9
7480 LOCATE 1,16
7500 LOCATE 1,18
7520 LOCATE 1,16
7550 LOCATE 1,18
8100 LOCATE 1,10
8170 LOCATE 1,20
8230 LOCATE 1,10
8250 LOCATE 1,20
```



TECNICAS DE ANALISIS

DISEÑO DE SISTEMAS

VAMOS a describir en las siguientes páginas la estructura general que suelen adoptar los diferentes documentos en que se plasma el diseño de un sistema informático. Naturalmente, la organización general e, incluso, el contenido de un dossier complejo de este estilo puede variar mucho dependiendo de los estándares que se sigan, del tipo de sistema a diseñar, o de las personas que lo redacten, pero vamos a presentar un esquema muy común de documentos y que, por tanto, será útil en la mayoría de los casos que se presenten.

Tal como se indica en el recuadro adjunto, se suelen incluir cuatro partes diferentes en el dossier de análisis del sistema correspondiente; desglosadas, a su vez, en varios apartados, hasta un total de diez epígrafes diferentes.

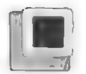
ESTRUCTURA DEL DISEÑO DE UN SISTEMA INFORMATICO

A. Información general.
   1. Datos generales del sistema.
   2. Presentación del sistema.
   3. Organigrama general.
   4. Organigramas de detalle.

B. Entradas y salidas.
   5. Diseño de documentos de salida.
   6. Diseño de los datos de entrada.

C. Estructura y procesos internos.
   7. Definición de cálculos.
   8. Tablas de decisiones lógicas.
   9. Datos interrelacionados e históricos.

D. Referencias y lista de contenido.
   10. Relación general de campos, referencias y datos utilizados.



Datos generales del sistema

Este documento normalmente es utilizado a modo de presentación e identificación del conjunto del sistema que se diseña. Suele contener los siguientes apartados:

a) Nombre del sistema o aplicación. Ha de ser un nombre conciso por el que se pueda aludir a la aplicación completa, pero lo suficientemente claro como para que a la vista del nombre las personas involucradas sepan de qué aplicación se trata. La identificación ha de ser lo más precisa posible, deslindándola de otras con ella relacionadas y sugiriendo el contenido exacto de los procesos que se realizan. Un nombre de dos o tres palabras suele ser lo adecuado. A veces se utiliza una estructura en etapas para estos nombres: por ejemplo, «Personal» para el sistema en general; «Personal. base de datos», «Personal, nóminas», «Personal, histórico», etc., para los diferentes procesos; «Personal, nóminas, datos variables», «Personal, nóminas, coeficientes», «Personal, nóminas, altas y bajas», etc., para los procesos de detalle.

b) Número de la aplicación. Un número o código alfanumérico (letras y números) para control e identificación automática. Caben dos posibilidades: numeración sucesiva de los diferentes sistemas y aplicaciones desarrollados en una organización o numeración en etapas (como se ha indicado anteriormente para el nombre literal) en la que se numeran con uno o dos dígitos (o letras) los diferentes sistemas y, dentro de cada uno, con otro u otros dígitos o letras los diferentes subsistemas, dentro de los cuales se numeran las diferentes aplicaciones, etc.

c) Número de revisión. Es usual tener que realizar sucesivos diseños de un mismo proceso. Conviene ir numerando sucesivamente estas versiones diferentes que del mismo sistema (y su documentación correspondiente) se van elaborando. A veces se utiliza una numeración decimal (sobre todo en documentos o sistemas que se actualizan con cierta frecuencia) utilizando los números enteros para las versiones que aportan cambios sustanciales y las cifras decimales para las pequeñas actualizaciones, obteniéndose una secuencia de revisiones del tipo: 2.1; 2.2; 2.3 (pequeños cambios); 3.1 (modificaciones importantes respecto de lo anterior); 3.2 (de nuevo cambios de poca importancia); 3.3..., etc.

d) Departamento. Se refiere al departamento de la empresa al que va destinado el sistema que se diseña. Dependiendo de la organización de la que se trate, en este u otros epígrafes equivalentes se incluirá unos datos u otros; el objetivo es, sencillamente, situar en la estructura de la empresa u organización, el sistema informático objeto de estudio.

DISEÑO DE SISTEMAS

Nombre ....

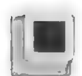
Número .... Revisión ....

Departamento ....

Equipo:	Preparado por:	Revisado por:	Aceptado por:
....	....	....	....
....	Fecha:....................	Fecha:....................	Fecha:....................
....	Firma:....................	Firma:....................	Firma:....................
....			

e) Equipo. Es útil referenciar el equipo físico (ordenador) sobre el que va a funcionar el sistema, si existen varios en la organización correspondiente. En este apartado se pueden incluir comentarios sobre los periféricos que se utilizarán o sobre algún dispositivo especial que se use en este sistema.

f) Control del trabajo de diseño. Es usual incluir varios casilleros donde se reseñen las personas que han diseñado y supervisado el sistema y la fecha en que se ha hecho. También, en ocasiones, se prevé una «aceptación» del diseño por parte del responsable del departamento que será usuario final del sistema una vez puesto en marcha.



Presentación del sistema

Descripción global del sistema donde se indiquen las ideas generales acerca del sistema que se diseña, su utilidad y finalidad, sus parámetros representativos, etcétera.

Deberían incluirse en esta breve presentación descriptiva, al menos, los siguientes puntos:

a) Clasificación de la aplicación. Tipo de aplicación de que se trate (genérica, particular, etc.) y finalidad a que se destina.

b) Objetivo. Actividad que se trata de informatizar. Problemas concretos a resolver.

c) Utilidad. Ventajas generales que se derivarán de la implementación del sistema que se diseña: menores costes, tiempos de proceso más cortos, más fiabilidad en los datos, obtención de datos o informaciones no disponibles por el sistema actual, etc.

d) Resultados a obtener. Informes que se derivarán de los procesos concebidos: contenido, finalidad, etc. Presentación global de estos informes poniendo el acento en su funcionalidad más que en su contenido o estructura formal.

e) Datos de entrada a partir de los cuales se realizarán los procesos previstos; su obtención, características generales. Como en el caso *d)*, más aspectos funcionales que de otro tipo.

f) Volúmenes representativos de la aplicación. Volúmenes de datos que se prevén, tiempos de proceso, periodicidad de la ejecución, etc.

g) Plan de puesta en marcha. Problemas que pueden aparecer en la implementación del sistema. Requisitos para la conversión de datos del sistema actual al propuesto o desde otras aplicaciones a la presente. Tareas a realizar para el arranque del sistema.

h) Plan de tiempos de la puesta en marcha. Cuantificación y planificación de tiempos para las actividades del apartado anterior.

i) Elementos hardware a utilizar. Configuración de equipo necesario (sobre todo si no es la configuración normal utilizada en otras aplicaciones).

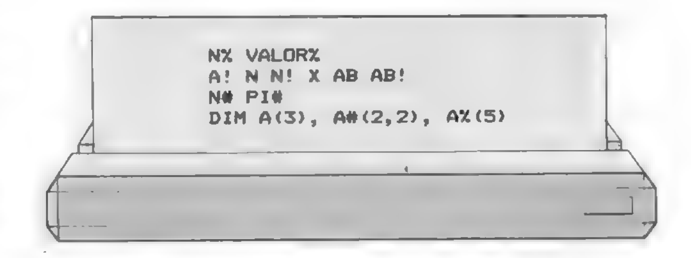
j) Posibilidades de expansión. Comentario sobre las previsiones actuales de modificación o expansión futura del sistema (si está previsto que se produzcan).

TECNICAS DE PROGRAMACION



CÓMO se distinguen las diferentes representaciones internas de los tipos de datos numéricos en los lenguajes de alto nivel? A menudo es preciso declarar explícitamente lo que deseamos que haga el ordenador con nuestros datos. En BASIC, por ejemplo, es el propio nombre de la variable el que decide cómo se van a representar sus valores en la memoria del ordenador. Así, si definimos una variable cuyo nombre termina en el carácter «%» (tanto por ciento) el intérprete de BASIC sabe automáticamente que esta variable va a tomar sólo valores enteros, y que deseamos que los guarde en la memoria como números binarios de dieciséis cifras. De igual manera, los nombres terminados en un signo de admiración indican que los valores pueden contener decimales, y que deben guardarse en la memoria en punto flotante, con una precisión normal, ocupando un total de cuatro octetos o bytes por cada valor que se les asigne. Por último, un nombre que termina en el signo «#» indica que se desean guardar los valores en punto flotante, pero con una precisión doble, ocupando un total de ocho octetos por cada valor. En condiciones normales, los nombres que no terminan en un carácter especial, sino en una letra corriente, se considerarán como variables cuyos valores se almacenan como números en punto flotante de precisión normal. Es decir, la ausencia de un carácter especial al final del nombre se considera equivalente a la presencia del signo de admiración.

Veamos algunos ejemplos:



```
N% VALOR%
A! N N! X AB AB!
N# PI#
DIM A(3), A#(2,2), A%(5)
```

Las variables de la primera línea adoptarán únicamente valores enteros, que ocuparán dos octetos. Las de la segunda línea tomarán valores con decimales, que se guardarán en memoria como números en punto flotante, de cuatro octetos. Las variables de la tercera línea tomarán también valores en punto flotante, pero con más cifras decimales, ocupando ocho octetos cada una. Finalmente, en la cuarta línea se definen variables de los tres tipos, pero que además son vectores o matrices. Obsérvese que la adición de un carácter especial al final de un nombre es suficiente para que se trate de una variable diferente, de manera que pueden existir al mismo tiempo las cuatro variables A, A!, A% y A#. La primera (A) queda definida como un vector de tres elementos en precisión normal, por lo que cada elemento ocupará cuatro octetos y los tres juntos un total de doce octetos. La segunda (A!) es un escalar (tiene un solo valor) en precisión simple y ocupará cuatro octetos. La tercera

(A%) es una matriz de valores enteros de dos filas y dos columnas, cuyos elementos ocupan dos octetos cada uno, lo que hace un total de ocho octetos. Finalmente la cuarta variable (A#) es un vector de cinco valores que se guardarán como números en punto flotante en precisión doble (ocho octetos cada uno), lo que hace un total de cuarenta octetos.

¿Y todo esto para qué sirve? ¿Por qué nos importa cuál sea la precisión de los valores, el número de los caracteres que ocupan, o si están en binario entero o en punto flotante? Pues bien: resulta que conocer la precisión en que se encuentran los datos es importante. En algunos casos, tendremos que hacer operaciones con decimales en las que nos interese que la precisión sea lo mayor posible. Tal vez la precisión normal (que equivale a unas siete cifras decimales significativas exactas) no baste para nuestros propósitos. En tal caso exigiremos que los cálculos se realicen con mayor precisión (precisión doble, que equivale a unas quince cifras decimales exactas) y añadiremos al final del nombre de nuestras variables el signo «#».

Otras veces sucede lo contrario. La precisión de las operaciones no es cuestión que nos interese demasiado, porque vamos a trabajar con números enteros pequeños, que pueden almacenarse con facilidad como binarios de dieciséis cifras (dos octetos). Ocurre que las operaciones realizadas con números binarios enteros suelen ser mucho más rápidas que las que se llevan a cabo en punto flotante, y las de precisión normal más rápidas que las de precisión doble. Por tanto, si la rapidez en los cálculos es nuestra primera consideración, procuraremos que todas nuestras variables sean enteras y les pondremos nombres terminados en el carácter «%».

Por último, podría ocurrir que parte de nuestras variables puedan tomar valores decimales durante la ejecución del programa, mientras que otras serán siempre enteras y otras exigirán mayor precisión en los cálculos. En tal caso, podemos utilizar el carácter de terminación adecuado para optimizar el programa y hacer que cada dato se encuentre en la memoria en la forma interna más adecuada para el buen funcionamiento del conjunto. Esto es especialmente útil si tenemos problemas de memoria disponible (es decir, si el programa es muy grande y los datos ocupan demasiado espacio). En tal caso puede ser absolutamente necesario reducir un poco el espacio ocupado por algunas variables (las que mantengan siempre valores enteros pequeños) para que el programa pueda llegar a funcionar.

Si nuestro programa va a utilizar al mismo tiempo numerosas variables enteras, en precisión normal o en precisión doble, puede llegar a ser molesto tener que escribir tantas veces el carácter final del nombre que indica al intérprete cuál es la representación interna elegida. Para evitarlo, algunos intérpretes de BASIC permiten al programador decidir que ciertas letras iniciales de nombre indiquen automáticamente el tipo de que se trata, siempre que la variable no termine en uno de los caracteres especiales, pues éstos tienen siempre precedencia. Veamos un ejemplo de aplicación de estas instrucciones declarativas, que deben situarse siempre al principio del programa:



La instrucción DEFINT N le dice al intérprete de BASIC que deseamos que toda variable cuyo nombre comience por N y no termine en uno de los caracteres especiales debe ser considerada como entera (*Integer* en inglés). En cuanto a la instrucción DEFβDBL A-C, indica que las variables que comiencen por A, B o C y no terminen en los caracteres especiales serán automáticamente variables en punto flotante en doble precisión. Si estas dos instrucciones hubieran aparecido en nuestro programas antes de la utilización de las variables que vimos más arriba, la representación interna de éstas habría cambiado de la siguiente manera: la variable N, que en ausencia de la instrucción DEFINT habría tenido valores en punto flotante en precisión normal, pasará a recibir valores binarios enteros. Las variables N#, N% y N! no se verán

afectadas. Y las variables AB y A, que de no haberse dado la instrucción DEFDBL habrían tenido también precisión normal, tendrán ahora precisión doble, ocupando un espacio doble en memoria del que antes requerían. Las variables A#, A%, A! y AB! no se verán afectadas.

En el lenguaje PASCAL, toda las variables que se utilicen en un programa deben ser definidas previamente en la sección declarativa. Por tanto, no existen reglas de declaración implícita del tipo, como en BASIC. Siempre hay que declarar explícitamente el tipo de cada una. Veamos, como ejemplo, la parte declarativa del programa de cálculo de interés compuesto que vimos en el capítulo 5:

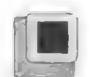
```
program INTERES;
(* Declaración de las variables *)
var
  nr, na, m, n: integer; (* contadores *)
  c:real; (* capital *)
  r, a: array[1..10] of integer;
                          (* réditos y años *)
  i: array[1..10,1..10] of real;
                        (* tabla de intereses *)
```

Se observará que el valor del capital (variable c) se ha declarado como variable «real» (esto es, sus valores se almacenarán como números en punto flotante). Esto se debe a que, en la mayor parte de los compiladores de PASCAL, los números enteros se representan internamente como binarios de dieciséis cifras, por lo que deben estar comprendidos entre −32768 y 32767. Como en este programa deseamos utilizar números mayores, nos vemos obligados a seleccionar el tipo real para el capital. Lo mismo ocurre, naturalmente, con los intereses (variable i), aunque en este caso tenemos además que disponerlos en forma de matriz de diez filas y diez columnas. En cambio, como los réditos y los tiempos de inversión (variables *r* y *a*) toman siempre valores pequeños, y como en este caso concreto sólo nos interesan réditos enteros, podemos definirlos como tales utilizando el tipo «integer». Si quisiéramos aplicar este programa a casos en que la tasa de interés (el rédito) pudiera ser fraccionario (por ejemplo, igual a 8,5), tendríamos que cambiar la parte declarativa del programa, definiendo como "real".

```
program INTERES;
(* Declaración de las variables *)
var
  nr, na, m, n: integer; (* contadores *)
  c:real; (* capital *)
  a: array[1..10] of integer; (* años *)
  r: array[1..10] of real; (* réditos *)
  i: array[1..10,1..10] of real;
                        (* tabla de intereses *)
```

En los intérpretes de APL existen tres formas principales de almacenamiento interno de los datos: las variables cuyos valores son todos iguales a cero o a uno se almacenan como conjuntos de números binarios de una sola cifra, empaquetados, por lo que ocupan tantos octetos como el cociente de dividir entre ocho el número de sus elementos (redondeado al entero inmediato superior). Las variables cuyos valores sean enteros comprendidos entre −32767 y 32767 se almacenarán en formato binario de dieciséis cifras, ocupando dos octetos por elemento. Por último, las restantes variables se guardan en punto flotante y en doble precisión, ocupando ocho octetos por elemento.

Sin embargo, el programador de APL no tiene que preocuparse por esto en absoluto. Pues, en efecto, el propio intérprete selecciona el formato adecuado para los datos en función de sus valores, y elige siempre aquél en que la variable ocupa el menor espacio posible. Además, si el valor de la variable o de alguno de sus elementos cambia durante la ejecución del programa, de manera que la representación interna que había tenido hasta ahora ya no es aplicable, el intérprete realiza automáticamente las conversiones adecuadas, sin que el programador tenga que ser consciente de ello. Por esta razón, se dice que en el lenguaje APL tan sólo hay dos tipos externos de datos: números y caracteres, lo que facilita considerablemente la programación eliminando además una fuente de considerable número de errores que afecta con frecuencia a otros lenguajes de programación.



Datos literales

Los datos literales son los que contienen información alfabética. En casi todos los lenguajes, los datos literales se intro-

ducen en los programas en forma de cadenas de caracteres encerrados entre comillas. En BASIC, por ejemplo, estas comillas tienen que ser dobles, y los caracteres incluidos en una cadena pueden contener cualquier cosa excepto la doble comilla. Esta, sin embargo, como cualquier otro carácter, se puede obtener de otra manera que ahora veremos.

Internamente, los caracteres se representan, como es natural, mediante números binarios. En los ordenadores modernos, un carácter se considera siempre equivalente a un octeto o byte, es decir, a ocho cifras binarias (ocho bits). Puede comprobarse con facilidad que con ocho dígitos binarios pueden escribirse exactamente 256 números distintos, que corresponden a los números decimales comprendidos entre 0 y 255. Por tanto, podremos definir 256 caracteres diferentes, a cada uno de los cuales les corresponderá una de esas representaciones internas.

Por supuesto, existen millones de representaciones internas posibles para los caracteres. Por ejemplo, podría haberse decidido que la letra «A» fuera internamente el número binario 00000001 (que corresponde al decimal 1) y la «B» estuviera representada por el binario 00101010 (que corresponde al decimal 42), o podría adoptarse otra equivalencia totalmente diferente. La tabla que hace corresponder a cada carácter un número binario (o el decimal equivalente) se llama «código».

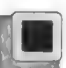
En la mayor parte de los ordenadores personales se ha adoptado el código ASCII, llamado así porque esas letras son las siglas de «American Standard Code for Information Interchange» (código patrón americano para intercambio de información). Este código define únicamente el significado de 94 caracteres representables, más el espacio en blanco y 33 de control dejando cierta libertad para la definición de los 128 caracteres restantes. La tabla siguiente presenta la parte común a todos los códigos ASCII de los distintos ordenadores personales. La primera columna contiene el código decimal equivalente al número binario que representa al carácter. La segunda columna es el carácter correspondiente. Los caracteres comprendidos entre los códigos 0 y 127 que no aparecen en la tabla son caracteres de control, no representables. Algunos se emplean para pasar página en las impresoras, o para otros menesteres parecidos.

CODIGO	CARACTER	CODIGO	CARACTER
8	Retroceso	78	N
10	Paso línea	79	O
13	Retorno	80	P
32	Espacio	81	Q
33	!	82	R
34	"	83	S
35	#	84	T
36	$	85	U
37	%	86	V
38	&	87	W
39	'	88	X
40	(	89	Y
41	)	90	Z
42	*	91	[
43	+	92	\
44	,	93	]
45	-	94	^
46	.	95	_
47	/	96	`
48	0	97	a
49	1	98	b
50	2	99	c
51	3	100	d
52	4	101	e
53	5	102	f
54	6	103	g
55	7	104	h
56	8	105	i
57	9	106	j
58	:	107	k
59	;	108	l
60	<	109	m
61	=	110	n
62	>	111	o
63	?	112	p
64	@	113	q
65	A	114	r
66	B	115	s
67	C	116	t
68	D	117	u
69	E	118	v
70	F	119	w
71	G	120	x
72	H	121	y
73	I	122	z
74	J	123	{
75	K	124	|
76	L	125	}
77	M	126	~

Pues bien: en BASIC existe una segunda manera de obtener caracteres, que consiste en utilizar la función CHR$, que aplicada a un número (el código del carácter) devuelve el carácter correspondiente. Por ejemplo, CHR$(78) es la letra N, y, por tanto, equivale totalmente a la cadena "N", expresada entre comillas dobles. Pero esta función puede emplearse en cualquier caso, incluso para obtener caracteres que no pueden ponerse entre comillas, como la doble comilla, CHR$(34), el retorno de carro, CHR$(13), así como cualquiera de los caracteres de control y de los que corresponden a la parte no común de los códigos ASCII, los comprendidos entre 128 y 255, que varían de ordenador en ordenador.

LOGO





Cómo repetir comandos

EN este momento, todos sabemos ya hacer un cuadrado. Los comandos podrían ser:



Como podemos ver, escribimos cuatro líneas de comandos que son exactamente iguales, es decir, repetimos lo mismo cuatro veces.

Para evitar el tener que escribir varias veces las mismas órdenes, la tortuga dispone del comando

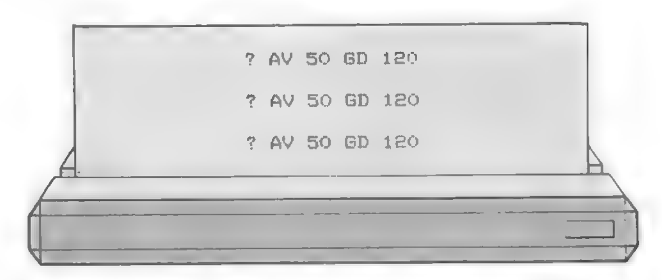
REPITE n (lista de comandos)

donde *n* indica el número de veces que queremos que la tortuga ejecute la lista de órdenes que ponemos entre corchetes.

Así, el dibujar el cuadrado resultará mucho más fácil. Nos basta con poner:



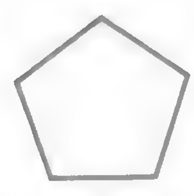
Vamos a probar ahora con un triángulo. El conjunto de comandos:



queda reducido a una sola orden si utilizamos el comando REPITE:



Si lo que queremos dibujar es un pentágono



escribiremos



No sabemos qué número de grados ha de girar la tortuga. Para conocerlo, vamos a usar un truco.

En cualquier figura que sea cerrada y cuyos lados no se cortan, la suma de todos sus ángulos vale 360 grados. Por tanto, para saber cuál es el valor de cada ángulo de un polígono regular tenemos que dividir el valor total (360) entre el número de ángulos. Es igual que cuando tenemos, por ejemplo, 100 caramelos y queremos repartirlos entre nuestros 5 amigos:

100 caramelos: cinco amigos = 20 caramelos para cada amigo

Con esto, ya podemos saber cuánto ha de girar la tortuga para pintar nuestro pentágono:

360 : 5 = 72

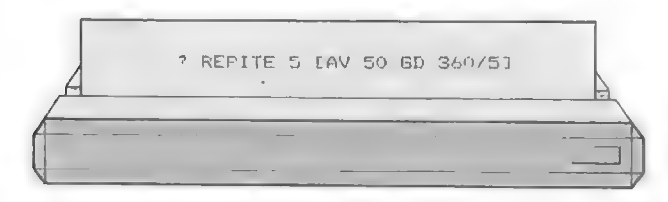
El comando queda así:



Lo que ocurre es que cada vez que queremos dibujar una figura de este tipo, primero tenemos que hacer una división para saber cuánto debe girar la tortuga.

Esto también lo podemos evitar haciendo que sea la propia tortuga la que calcule este valor, pero ¿cómo? Pues muy fácil. Nuestra tortuga sabe sumar, restar, multiplicar y dividir. Por tanto, podemos decirle que haga ella la división.

Teniendo esto en cuenta, la orden que haría que la tortuga dibujara el pentágono sería:

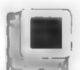


Como vemos, la división se representa mediante la barra inclinada (/).

En el caso de que quisiéramos dibujar un triángulo, pondríamos:



La división de 360 entre 3 da 120. Por ello, al querer pintar un triángulo en otras ocasiones hemos hecho que la tortuga girase este número de grados y no 60.

 Una fórmula general para polígonos

Después de haber dibujado varios polígonos (triángulo, cuadrado, pentágono) podemos ver que para todos el comando REPITE es muy parecido:

triángulo:

REPITE 3 (AV 50 GD 360/3)

cuadrado:

REPITE 4 (AV 50 GD 360/4)

pentágono:

REPITE 5 (AV 50 GD 360/5)

En todos hacemos lo siguiente:

— Primero, ponemos REPITE.

— Luego, un número que indica el número de lados (o de ángulos, ya que coinciden) que tiene la figura.

— En la lista de órdenes (entre corchetes) ponemos:

— Avanzar un número de pasos.

— Girar una cantidad que se obtiene dividiendo 360 entre el número de ángulos.

Por tanto, en general, cuando queramos dibujar cualquier polígono, el comando será así:

REPITE número-lados (AV longitud GD 360/número-lados)

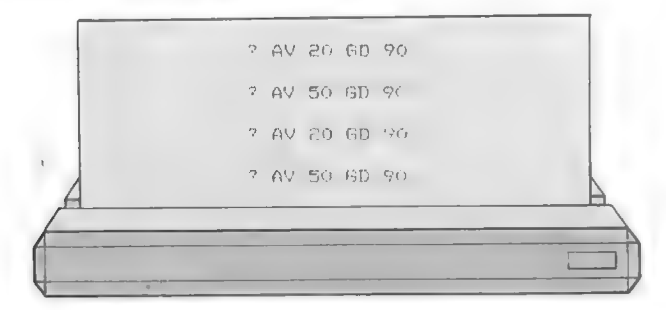
Con esto ya podemos pintar polígonos que tengan distinto número de lados y, además, sin necesidad de tener que escribir muchos comandos: solamente uno.

 Os proponemos

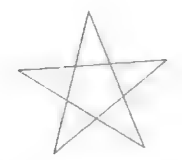
1. Utilizando la fórmula general puedes pintar diferentes polígonos. Por ejemplo:

hexágono :	6 lados
heptágono :	7 lados
octógono :	8 lados
eneágono :	9 lados
decágono :	10 lados
endecágono :	11 lados
dodecágono :	12 lados

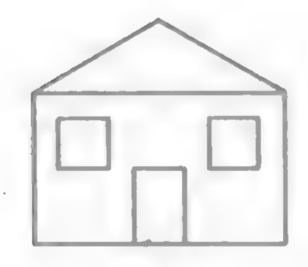
2. Intenta dibujar un rectángulo usando el comando REPITE. Es muy fácil, fíjate bien cómo lo haces sin utilizar este comando. Por ejemplo:



3. Pinta esta estrella con el comando REPITE. Para ello, sustituye en la fórmula general para pintar polígonos el número de lados por el número de puntas y 360 por 720.



4. Dibuja esta casa.



Circunferencias

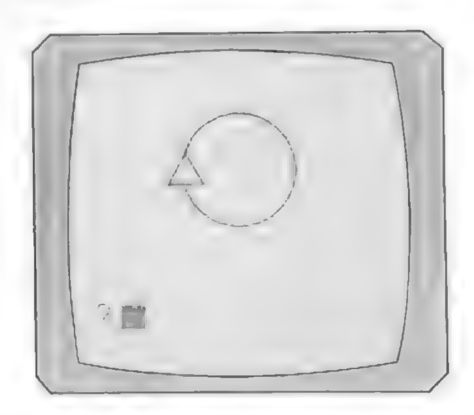
Al dibujar los diferentes polígonos, podemos ver que según aumenta el número de lados, la cantidad de grados que la tortuga va girando es cada vez más pequeña. Esto es lo mismo que si tenemos los 100 caramelos de antes, pero cada vez queremos repartirlos entre más amigos.

Además, al aparecer cada polígono en la pantalla, conforme aumenta el número de lados, la figura se va pareciendo más a una circunferencia.

Por tanto, podemos dibujar una circunferencia usando el comando REPITE. Para ello, la tortuga tiene que girar muy poquito cada vez y la figura ha de tener un número de lados muy grande, en concreto 360. Entonces, si ponemos:



nos queda:

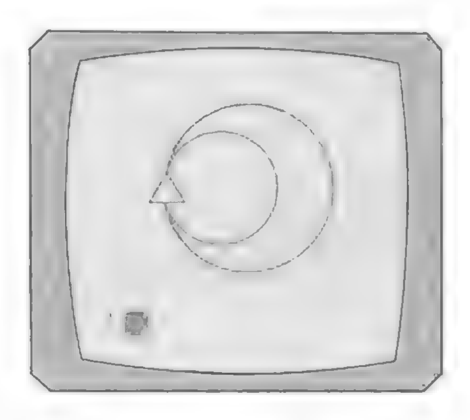


Para hacer una circunferencia más grande, tenemos que aumentar el número de pasos del comando AV.

Así, si escribimos:



nos quedará, si no hemos borrado la circunferencia anterior:



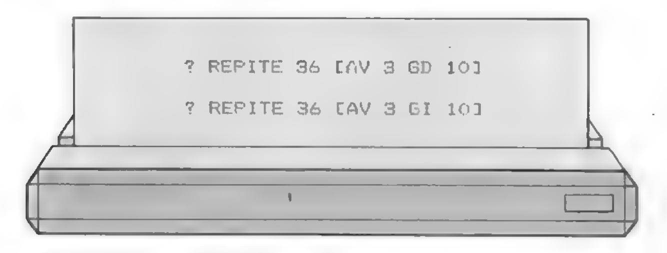
Lo que ocurre es que esta orden resulta muy pesada para la tortuga y tarda bastante en ejecutarla, ya que tiene que hacer 360 veces lo que hemos puesto entre corchetes.

Si escribimos la orden de esta otra manera:

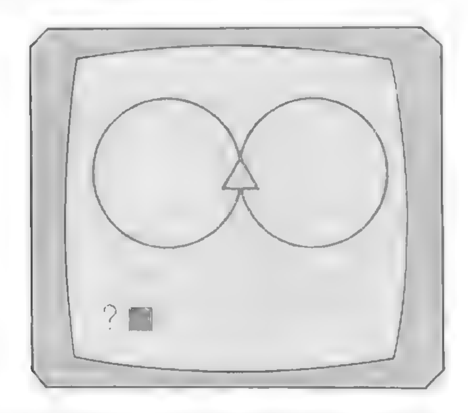


obtenemos el mismo resultado (una circunferencia), pues lo único que hacemos es quitarle un cero (0) al 360 (nos queda 36) y ponérselo al 1 (nos queda 10). Además, la tortuga tarda menos en ejecutarla.

Vamos a dibujar dos circunferencias. Si ponemos los comandos:

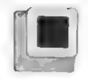


nos queda:



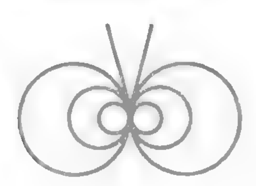
En resumen, para dibujar circunferencias hemos de poner:

REPITE 36 (AV tamaño GD 10)
o
REPITE 36 (AV tamaño GI 10)

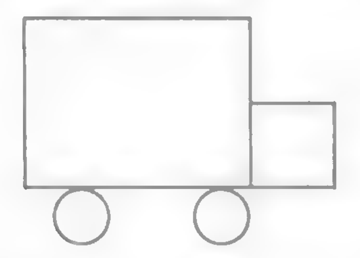


Os proponemos

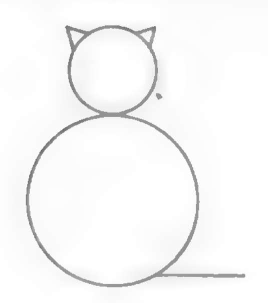
1. Pinta esta mariposa.



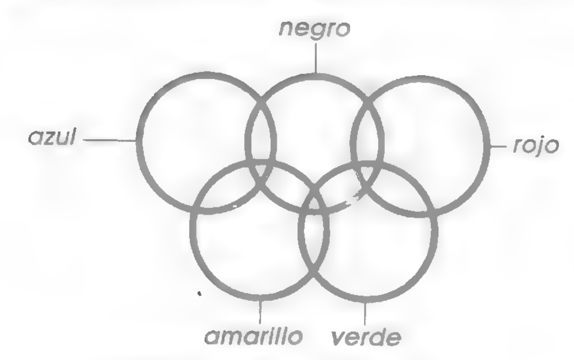
2. Puedes dibujar un camión.



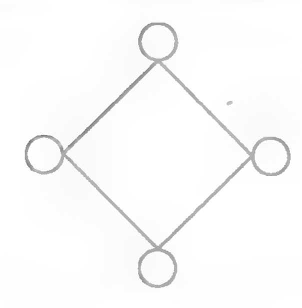
3. También puedes hacer un gato.



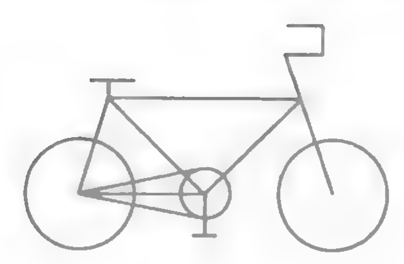
4. Intenta pintar los aros olímpicos, usando distinto color para cada círculo.



5. Atrévete con esta figura.

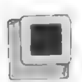


6. ¿Qué te parece esta bicicleta?



PASCAL



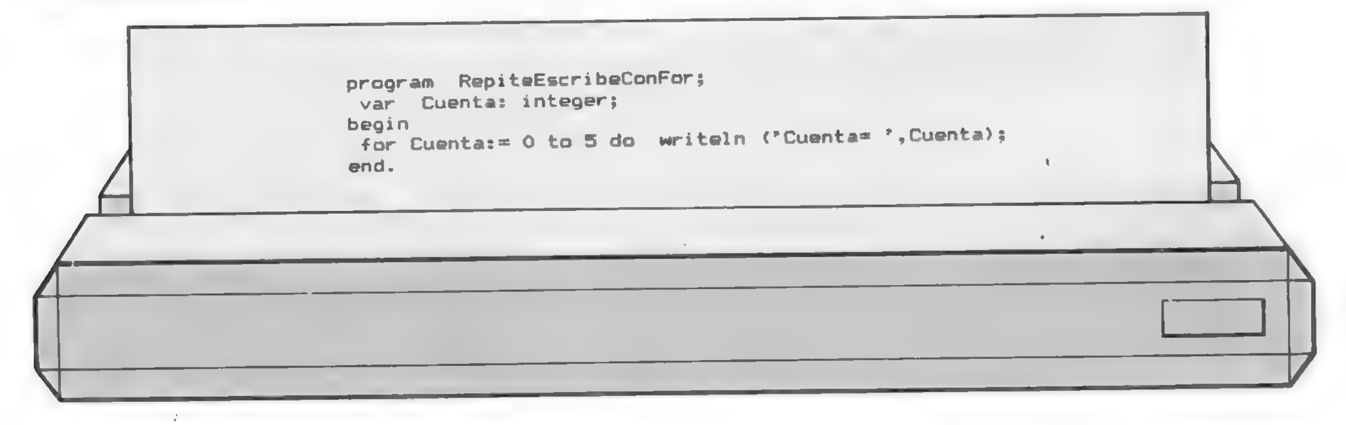


La estructura FOR

EN el programa RepiteEscribe que vimos al estudiar la estructura REPEAT se ejecutaban las instrucciones del bucle un número preestablecido de veces utilizando la variable Cuenta como "contador de programa".

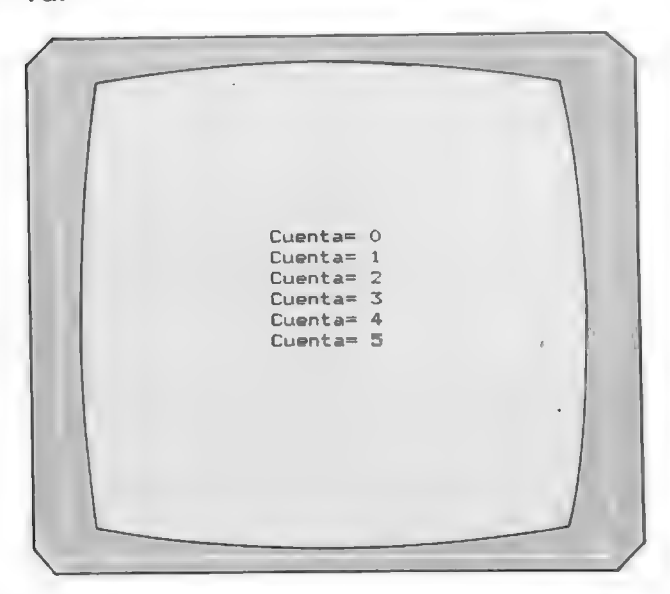
Es tan habitual tener que repetir instrucciones o bloques de instrucciones un número fijo de veces, que el PASCAL dispone de una estructura específica para ello: la estructura FOR.

Con ella, el programa equivalente a RepiteEscribe podría quedar:



```
program  RepiteEscribeConFor;
 var  Cuenta: integer;
begin
 for Cuenta:= 0 to 5 do  writeln ('Cuenta= ',Cuenta);
end.
```

Nuevamente, en la pantalla aparecerá:



La estructura es la siguiente:

FOR (contador:= primer valor) TO (último valor) DO (instrucción)

Tras la palabra reservada FOR se escribe la expresión de asignación del primer valor al contador de programa (Cuenta:= 0); tras ello siguen la palabra reservada TO, el último valor que tomará el contador (5), la palabra DO y, por fin, la instrucción a repetir que, como siempre, puede ser simple o estructurada.

Al ejecutarse el programa, cuando se llega a una estructura FOR, al contador de programa se le asigna el primer valor. Tras ello se ejecuta la instrucción y el contador pasa a tomar el siguiente valor de manera automática, volviéndose a ejecutar la instrucción nuevamente y a

variar el contador, etc., hasta que, por fin, la instrucción se ejecuta teniendo el contador el valor especificado como último, momento tras el cual se pasa a ejecutar lo siguiente a la estructura.

Traduciendo del inglés:

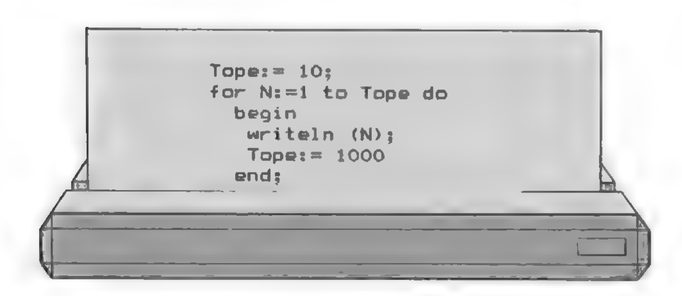
```
FOR          Cuenta:= 0
Para   Cuenta valiendo desde 0

 TO      5     DO     writeln...
hasta    5    hacer   writeln...
```

Para especificar los valores inicial y final se puede utilizar cualquier expresión que proporcione un resultado del mismo tipo que la variable utilizada como contador.

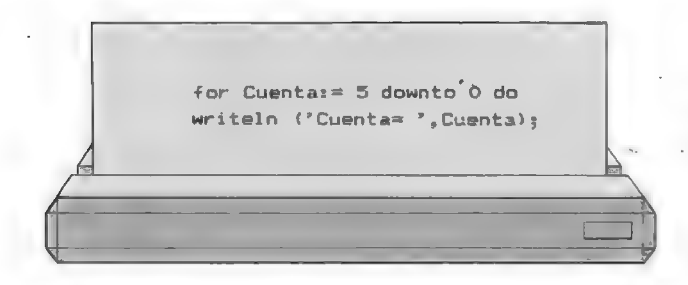
Aspectos importantes:

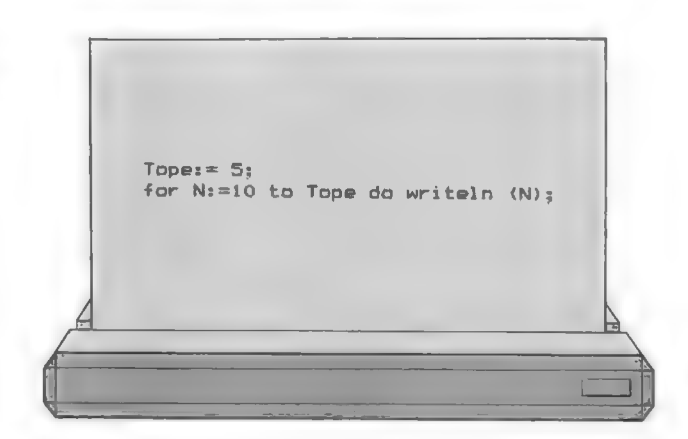
— El valor final se calcula nada más llegar a la estructura FOR; por ello, si alguno de los elementos de la segunda expresión cambiara durante alguna de las repeticiones del bucle, el valor final no se vería afectado. Por ejemplo, si *N* y Tope son variables INTEGER:



Tope pasará de valer 10 a valer 1000, pero sólo se sacarán los valores de *N* hasta llegar a 10, que era el valor de Tope cuando se llegó a la estructura FOR.

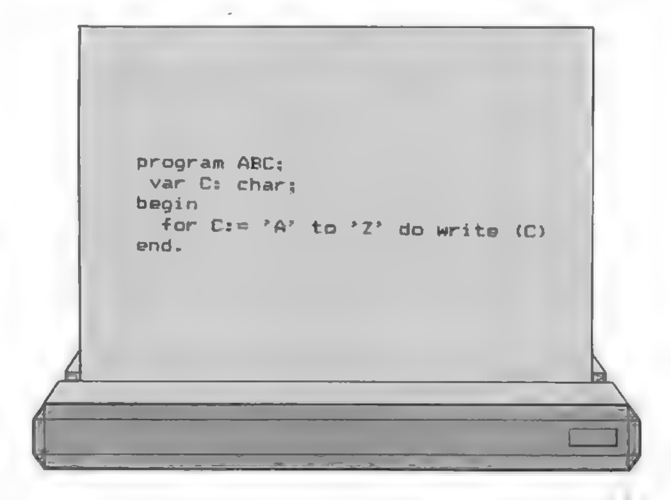
— Si el valor final fuese anterior al inicial, no se ejecutarían ni una vez las instrucciones del bucle:





así, N no se llega a escribir nunca (es decir, la comprobación de llegada al valor final se hace antes de ejecutarse la instrucción).

La variable de control no puede ser de cualquier tipo. Sólo el tipo INTEGER y aquellos otros que tienen asociados números ordinales son admisibles (junto con sus posibles subrangos), es decir, aquellos tipos cuyos posibles valores están ordenados y con los que se pueden usar las funciones PRED, SUCC y ORD. Por ahora sólo hemos visto el tipo CHAR (y el BOOLEAN). Podríamos, por tanto, escribir:



Al correr el programa saldrá por la pantalla:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Existe una forma alternativa de la estructura FOR para cuando se desea que la variable de control vaya cambiando de valor en orden inverso al normal. Consiste en usar la palabra reservada DOWNTO en lugar de TO.

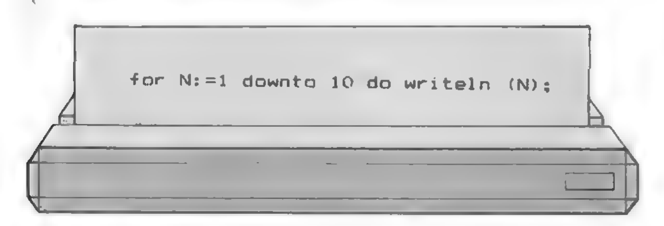
En el programa RepiteEscribeConFor podríamos poner:

Esta vez, en la pantalla aparecerían los números en orden inverso al anterior.

La traducción del inglés sería:

```
 FOR          Cuenta:= 5            DOWNTO
Para Cuenta valiendo desde 5 para abajo hasta
                         0   DO  ...
                         0  hacer...
```

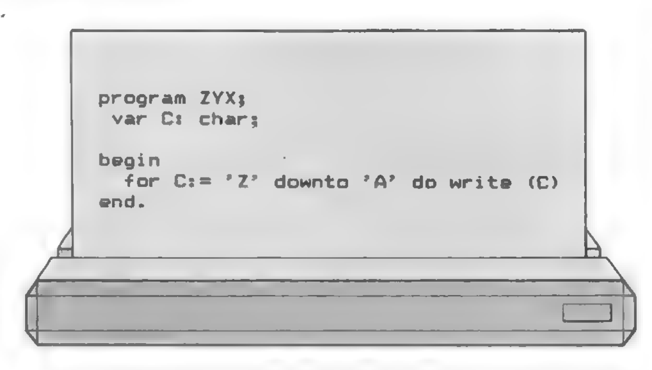
El segundo aspecto antes mencionado es aquí el inverso. Con:



```
for N:=1 downto 10 do writeln (N);
```

no se llegaría a escribir nada.

Igualmente podríamos hacer con el tipo CHAR:



```
program ZYX;
 var C: char;

begin
  for C:= 'Z' downto 'A' do write (C)
end.
```

La estructura FOR es a su vez una instrucción estructurada, y, por tanto, podría formar parte de la instrucción a repetir en otra estructura FOR:

```
program DobleFor;
(* Este programa saca todas las casillas
   del tablero del juego de los barquitos *)

const
  UltimaFila    = 'J';
  UltimaColumna = 10;

var
  Fila    : 'A'..UltimaFila;
  Columna: 1..UltimaColumna;

begin
 for Fila:= 'A' to UltimaFila do
  (* Para cada fila empezando por la A,
     sacar todas sus casillas:          *)

   begin
    writeln;
    writeln ('En la fila ',Fila,' tenemos: ');
    for Columna:= 1 to UltimaColumna do
        write (Fila,Columna,' ');
    writeln
   end
end.
```

Por último, dos advertencias y un consejo dirigidos fundamentalmente a los expertos en BASIC:

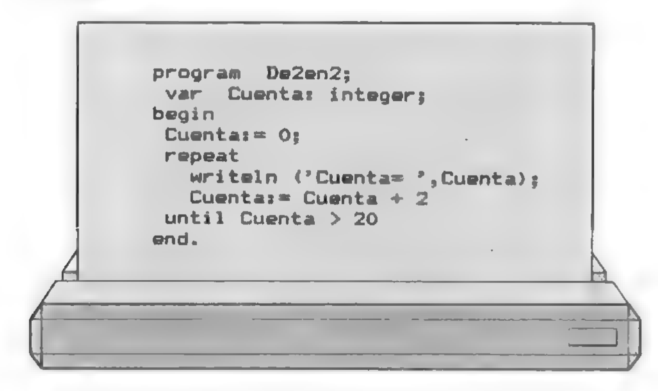
— El valor del contador de programa al acabar de ejecutarse la estructura FOR puede que sea el siguiente al especificado como final, pero puede que no; depende de cada compilador en concreto.

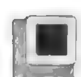
— No se debe alterar el valor de la variable de control mediante ninguna instrucción dentro del bucle: podrían suceder cosas imprevistas. Por tanto, No se deben hacer cosas como:



```
for N:=-100 to 100 do
  begin
    ... ;
    N:=117
  end;
```

— El único tipo de contador numérico admitido es el INTEGER, y además sólo se admiten incrementos de 1 (o de −1 con DOWNTO). Por ello, lo mejor para cuando se desean variaciones distintas de la unidad es utilizar una estructura REPEAT o WHILE:





Diseño de un programa

Ahora que ya conocemos las principales estructuras de control del PASCAL, vamos a desarrollar un programa que utilice algunas de ellas.

Este programa va a servir para dibujar en la pantalla tres tipos de figuras: cuadrados, pirámides y rombos. Un boceto del programa podría ser:

1. Preguntar qué tipo de figura se desea.
2. Preguntar de qué tamaño.
3. Dibujar la figura deseada.

Para poder dibujar varias figuras sin tener que arrancar el programa cada vez, podríamos cambiar su estructura a la siguiente:

Repetir lo siguiente...

1. Borrar la pantalla.
2. Preguntar qué tipo de figura se desea.
3. Preguntar de qué tamaño.
4. Dibujar la figura deseada.
5. Preguntar si se desea seguir.

...hasta que no se desee seguir.

Ya hemos visto sobrados ejemplos sobre cómo hacer los apartados 1, 2, 3 y 5. Sin embargo, el punto 4 es más complejo; podría ser algo así:

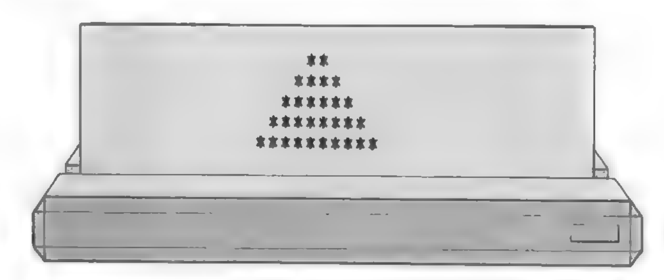
— Si se desea un cuadrado entonces: Pintar un cuadrado.

Pero si no, entonces:

— si es una pirámide lo deseado: Pintar una pirámide.

y si tampoco es una pirámide: Pintar un rombo.

Una pirámide podría tener el siguiente aspecto:



Es decir, una pirámide está formada por una serie de filas que constan cada una, en primer lugar, de una secuencia de espacios en blanco, que es más corta cuanto más baja sea la fila; a continuación viene una secuencia de asteriscos que, por el contrario, va haciéndose más larga.

Un rombo podemos suponer que está formado por dos pirámides, puesta una contra la otra.

Por último, un cuadrado es simplemente un número dado de filas con una cantidad constante de asteriscos.

Podemos pasar ya a escribir el programa:

```
program Dibujos;

const
 TotalMaximo = 20; TotalMinimo = 4;
var
 C, Tipo               : char;
 Fila, Columna, Total: integer;
 OK                    : boolean;

begin
repeat (* repetir todo *)

 ClrScr;  (* esto borra la pantalla *)
(* Si hace falta, cambiarlo por PAGE, o
por lo que necesite nuestro compilador: *)
```

```
write ('¿C(uadrado),R(ombo),P(irámide)? ');
repeat
 readln (Tipo);
 OK:=(Tipo='C') or (Tipo='R') or (Tipo='P');
 if not OK then writeln ('No vale, repita.')
until OK;

write ('¿Tamaño? ');
repeat
 readln (Total);
 OK:= false;(* se supone mala respuesta a priori *)
 if Total > TotalMaximo then
     writeln ('Demasiado, repita.')
   else
     (*-----------------------*)
     if Total < TotalMinimo then
         writeln ('Demasiado poco, repita.')
       else
         OK:= true (* se da por bueno *)
     (*-----------------------*)
 until OK;

 (*==========================================*)
if Tipo='C' then
     (*------------------------------*)
     (*      Pintar cuadrado:        *)
     (*------------------------------*)
     for Fila:=1 to Total do
       begin
        for Columna:=1 to Total do write('*');
        (* tras cada fila, saltar a la siguiente *)
        writeln
       end
     (*------------------------------*)
 else
   if Tipo='P' then
     (*------------------------------*)
     (*    Pintar pirámide:          *)
     (*------------------------------*)
     for Fila:= 1 to Total do
       begin
         for Columna:= 1 to Total-Fila do
           write(' ');
         for Columna:= 1 to 2*Fila do write('*');
         writeln
       end
     (*------------------------------*)

    else
     (*------------------------------*)
     (*      Pintar rombo:           *)
     (*------------------------------*)
     begin
      (* Pintar una pirámide bajita: *)
      for Fila:= 1 to Total div 2 do
       begin
        for Columna:= 1 to Total div 2 - Fila do
          write(' ');
        for Columna:= 1 to 2*Fila do write('*');
        writeln
       end;

      (* Pintar otra pirámide invertida: *)
      for Fila:= 1 to Total div 2 do
       begin
        for Columna:= 1 to Fila do  write(' ');
        for Columna:= 1 to Total-2*Fila do
          write('*');
        writeln
       end
     end;
 (*==========================================*)
 write ('¿Sigo? (S/N) '); readln (C);

until C='N'
end.
```

El desarrollo que se ha hecho de este programa, descomponiéndolo en tareas más sencillas, éstas a su vez en otras, y así hasta llegar a unas que sean fácilmente convertibles en instrucciones de programa, hace que la posibilidad de error disminuya mucho. Más adelante veremos que esto se pude sistematizar en PASCAL de una menera muy cómoda mediante el empleo de "procedimientos".

OTROS LENGUAJES

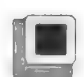
SISTEMAS OPERATIVOS: UNIX



UNIX es un sistema operativo multiusuario (de tiempo compartido o "time sharing"). Esto, ¿qué quiere decir? Pues sencillamente que UNIX permite a varios usuarios compartir los recursos del ordenador, dando la apariencia de que tienen un equipo para cada uno. La explicación "técnica" de este fenómeno es bien simple: el sistema operativo hace que el procesador central de la máquina atienda a cada usuario durante un determinado intervalo de tiempo. Como los equipos cada vez son más rápidos, los usuarios no perciben que realmente sólo disponen del ordenador para ellos solos durante esos pequeños intervalos, dando así la apariencia de continuidad.

Históricamente, Unix es un producto que salió de los laboratorios Bell (pertenecientes a la compañía AT&T). Su primera versión fue escrita por Ken Thompson entre 1968 y 1970 para un miniordenador PDP-7 de Digital Equipment Corporation. Al poco tiempo se le unió Dennis Richie (creador del lenguaje de programación C), junto con el cual escribió una nueva versión para los PDP-11, y fue en 1973 cuando reescribían UNIX en lenguaje C, también para los PDP-11.

En 1978 UNIX se comercializa por primera vez con el nombre de UNIX Version 7. Con el paso del tiempo, el sistema va mejorando e incorporando nuevas utilidades, razón por la cual van surgiendo, lógicamente, nuevas versiones. Así, en 1982 aparece la versión UNIX System III, y en 1983 UNIX System V. Al margen de estas versiones, todas ellas originadas en la propia AT&T, han aparecido otras provenientes de otras companías o centros de investigación, entre las cuales es de destacar XENIX (de Microsoft), y las series BSD de la Universidad de Berkeley.



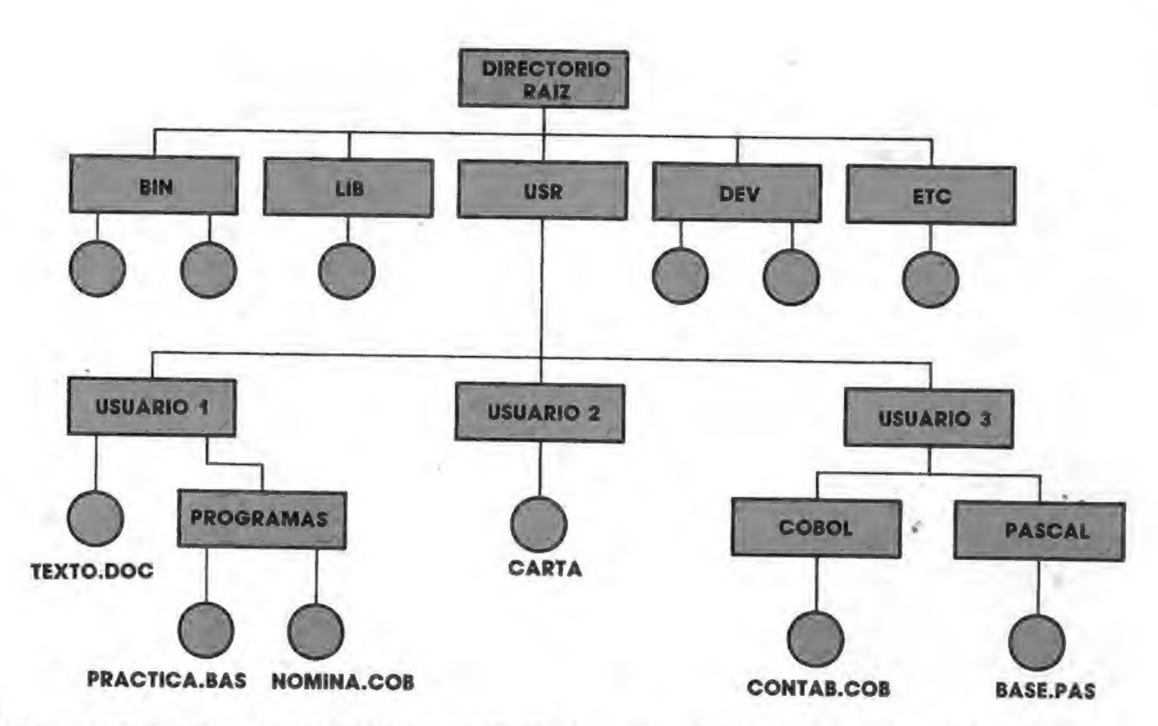
Características generales

Las características más relevantes de UNIX son las siguientes:

— A diferencia de la mayoría de los sistemas operativos, que están escritos en el lenguaje ensamblador de la máquina a la que van destinados, UNIX está escrito en un lenguaje de alto nivel como es el C, de ahí su alto grado de portabilidad, es decir, la facilidad de ejecutar los programas escritos sobre UNIX sobre distintas máquinas, sin tener que hacer cambios en los mismos.

— Tiene una estructura de ficheros jerarquizada, es decir, en forma de árbol invertido, el cual nace de un directorio raíz que contine ficheros y subdirectorios, los cuales, a su vez, contienen otros ficheros y/o subdirectorios, creándose así una estructura ramificada "hacia abajo".

— La interface con el usuario es fácilmente modificable. Esta interface la compone un intérprete de comandos (la "shell"), que además dispone de su propio "lenguaje de programación". El decir que es fácilmente modificable significa que se podría adaptar para que fuera



Esquema del sistema de ficheros de UNIX. Los directorios están representados por rectángulos, y los ficheros por círculos.

guiando al usuario en las acciones a realizar, de manera que la complicada sintaxis de UNIX fuera transparente para él.

— Se puede redireccionar la entrada y la salida. Esto significa que el resultado de la ejecución de un programa o de un comando (que en el fondo no es más que otro programa), se puede llevar a cualquier sitio, ya sea un fichero en disco como un dispositivo periférico (por ejemplo, la pantalla).

— Es un sistema operativo multitarea, lo cual permite a cada usuario ejecutar distintos trabajos con aparente simultaneidad.

— Permite trabajar en modo multiusuario, o lo que es lo mismo, tiene la posibilidad de tener activados simultáneamente varios puestos de trabajo.

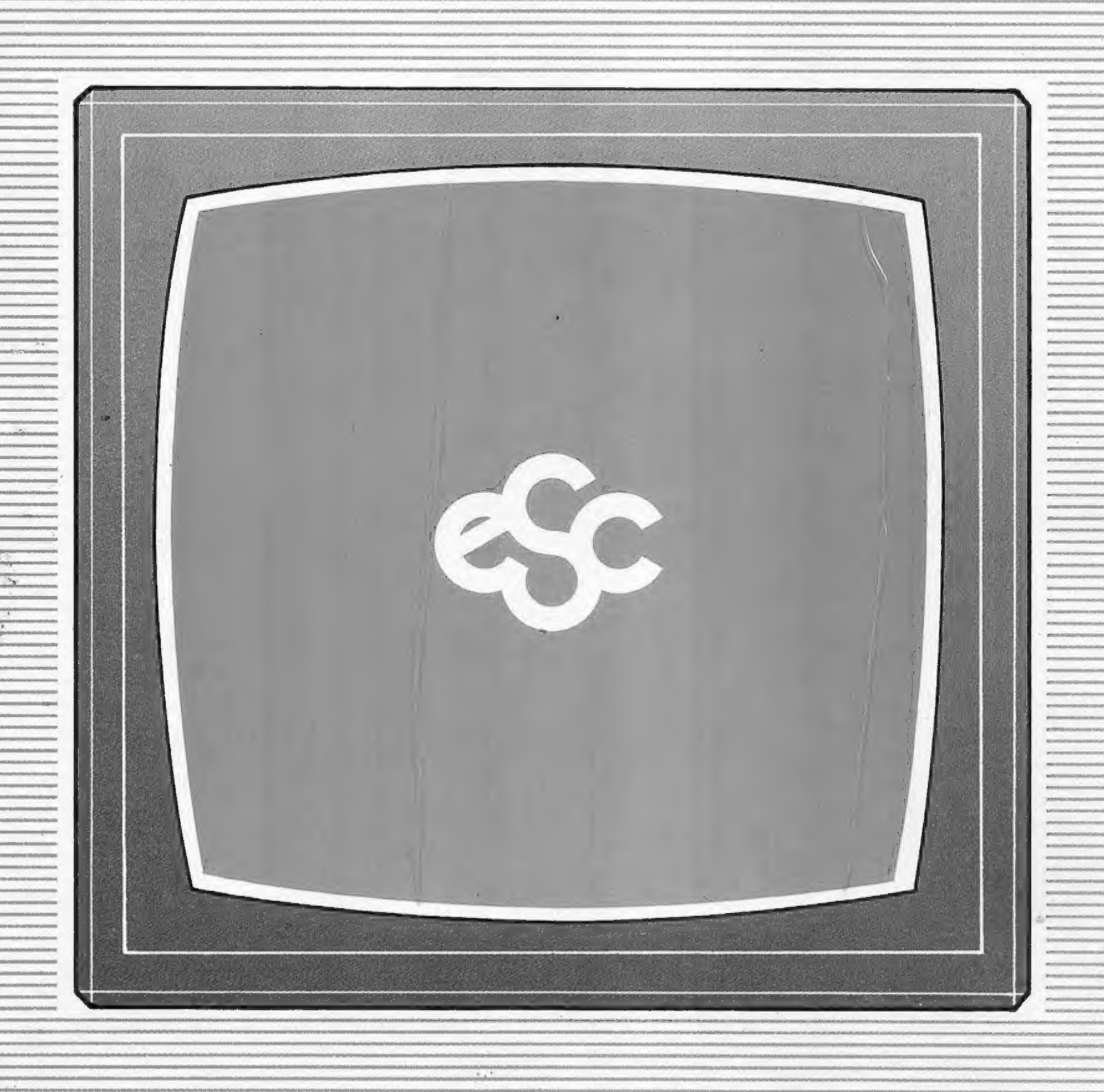
— Se pueden comunicar distintos procesos mediante "pipe-lines" (tuberías), de manera que la salida de un proceso puede ser la entrada de otro, y la salida de éste la entrada de un tercero... y así sucesivamente.

— Permite el intercambio de mensajes entre usuarios, ya sea por medio de "buzones de cartas", o mediante comunicación interactiva entre ellos.

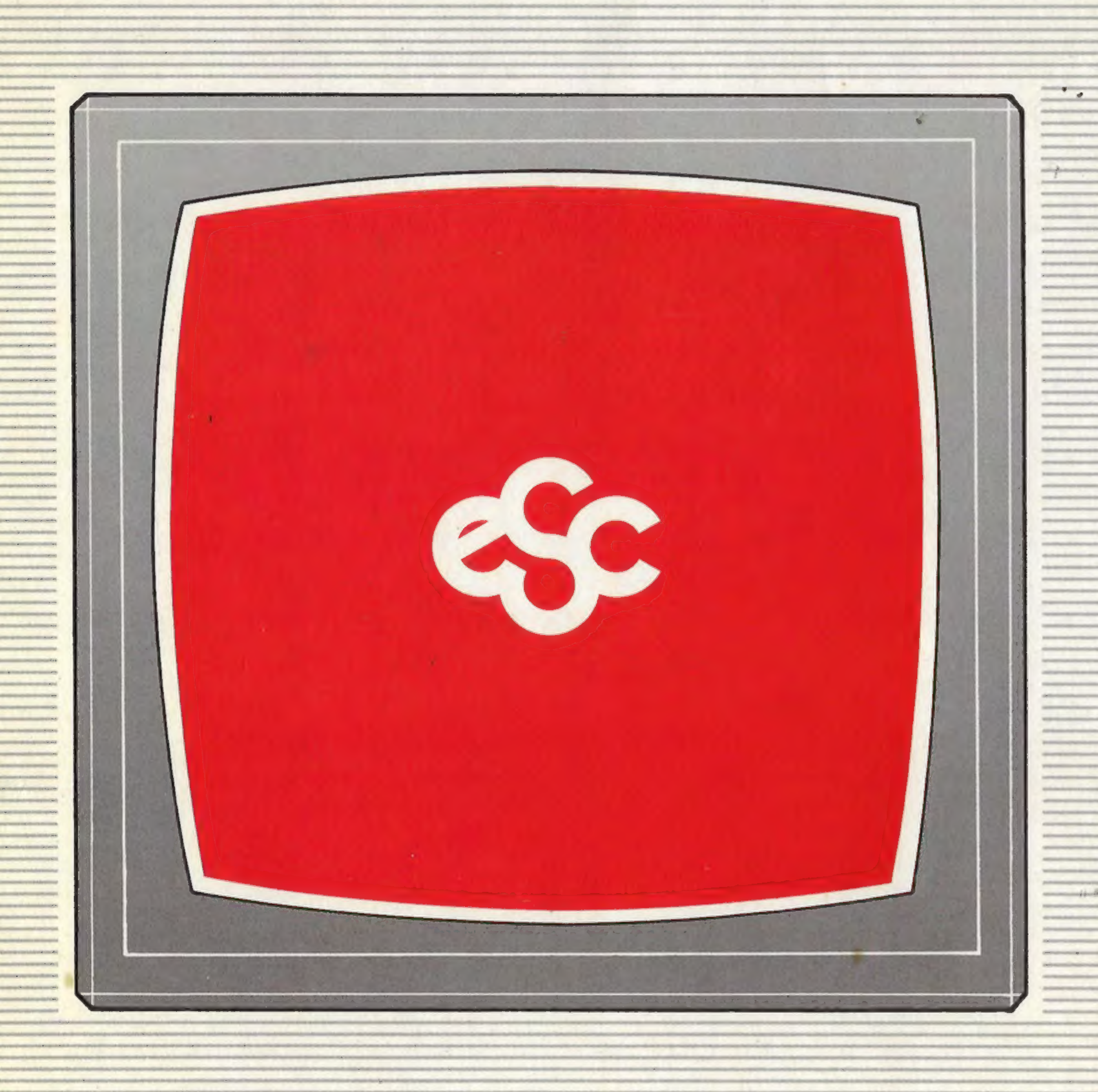
No hay que perder de vista, de todas formas, algunos de los inconvenientes de UNIX. Por una parte, su mayor potencia implica una mayor dificultad en su aprendizaje (muy superior a la de otros sistemas operativos como CP/M o MS/DOS). Esto es debido en gran parte al elevado número de comandos de que dispone, y a la engorrosa y complicada sintaxis de los mismos, lo que hace que UNIX no sea muy apto para los principiantes. Otro factor que hay que tener en cuenta es que el elevado número de versiones, todas ellas con mayor o menor grado de incompatibilidad, podrían hacer difícil su estandarización, aunque la compañía creadora de UNIX, AT&T, ha dejado clara su intención de hacer del UNIX System V un estándar del mercado.



Estructura general del sistema operativo UNIX.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼